

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re application of : THE COMMISSIONER IS AUTHORIZED  
Hajime OGAWA et al. : TO CHARGE ANY DEFICIENCY IN THE  
Serial No. NEW : FEES FOR THIS PAPER TO DEPOSIT  
Filed June 30, 2003 : ACCOUNT NO. 23-0975  
: Attn: APPLICATION BRANCH  
: Attorney Docket No. 2003\_0866A

COMPILER APPARATUS WITH FLEXIBLE  
OPTIMIZATION

---

CLAIM OF PRIORITY UNDER 35 USC 119

Commissioner for Patents  
P.O. Box 1450  
Alexandria, VA 22313-1450


Sir:

Applicants in the above-entitled application hereby claim the date of priority under the International Convention of Japanese Patent Application No. 2002-195305, filed July 3, 2002, as acknowledged in the Declaration of this application.

A certified copy of said Japanese Patent Application is submitted herewith.

Respectfully submitted,

Hajime OGAWA et al.

By   
Michael S. Huppert  
Registration No. 40,268  
Attorney for Applicants

MSH/kjf  
Washington, D.C. 20006-1021  
Telephone (202) 721-8200  
Facsimile (202) 721-8250  
June 30, 2003

日 本 国 特 許 庁  
JAPAN PATENT OFFICE

別紙添付の書類に記載されている事項は下記の出願書類に記載されている事項と同一であることを証明する。

This is to certify that the annexed is a true copy of the following application as filed with this Office

出 願 年 月 日

Date of Application:

2002年 7月 3日

出 願 番 号

Application Number:

特願2002-195305

[ ST.10/C ]:

[ JP2002-195305 ]

出 願 人

Applicant(s):

松下電器産業株式会社

2003年 2月21日

特許庁長官  
Commissioner,  
Japan Patent Office

太田信一郎

出証番号 出証特2003-3009821

【書類名】 特許願

【整理番号】 5037740105

【あて先】 特許庁長官殿

【国際特許分類】 G06F 9/45

【発明者】

【住所又は居所】 大阪府門真市大字門真 1 0 0 6 番地 松下電器産業株式会社内

【氏名】 小川 一

【発明者】

【住所又は居所】 大阪府門真市大字門真 1 0 0 6 番地 松下電器産業株式会社内

【氏名】 瓶子 岳人

【発明者】

【住所又は居所】 大阪府門真市大字門真 1 0 0 6 番地 松下電器産業株式会社内

【氏名】 坂田 俊幸

【発明者】

【住所又は居所】 大阪府門真市大字門真 1 0 0 6 番地 松下電器産業株式会社内

【氏名】 高山 秀一

【発明者】

【住所又は居所】 大阪府門真市大字門真 1 0 0 6 番地 松下電器産業株式会社内

【氏名】 道本 昌平

【発明者】

【住所又は居所】 大阪府門真市大字門真 1 0 0 6 番地 松下電器産業株式会社内

【氏名】 濱田 智雄

【発明者】

【住所又は居所】 大阪府門真市大字門真 1 0 0 6 番地 松下電器産業株式会社内

【氏名】 宮地 涼子

【特許出願人】

【識別番号】 000005821

【氏名又は名称】 松下電器産業株式会社

【代理人】

【識別番号】 100109210

【弁理士】

【氏名又は名称】 新居 広守

【電話番号】 06-4806-7530

【手数料の表示】

【予納台帳番号】 049515

【納付金額】 21,000円

【提出物件の目録】

【物件名】 明細書 1

【物件名】 図面 1

【物件名】 要約書 1

【プルーフの要否】 要

【書類名】 明細書

【発明の名称】 コンパイラ装置

【特許請求の範囲】

【請求項 1】 ソースプログラムを機械語プログラムに翻訳するコンパイラ装置であって、

生成する機械語プログラムを最適化する旨の指示を取得する指示取得手段と、  
取得された指示に従って機械語命令列を生成することにより、最適化を施す最適化手段とを備え、

前記最適化手段は、前記指示取得手段がグローバルメモリ領域に配置する配列データについての指示を取得した場合には、取得した指示に従ってグローバルメモリ領域に配置する配列データを決定することにより、前記最適化を施すことを特徴とするコンパイラ装置。

【請求項 2】 前記指示取得手段は、前記ソースプログラムを翻訳する旨の指示とともに、グローバルメモリ領域に配置する配列データの最大データサイズについての指定を取得し、

前記最適化手段は、前記ソースプログラムで宣言された配列データのうち、前記最大データサイズを超えないデータサイズの配列データについてはグローバルメモリ領域に配置し、前記最大データサイズを超えるデータサイズの配列データについてはグローバルメモリ領域外のメモリ領域に配置する

ことを特徴とする請求項 1 記載のコンパイラ装置。

【請求項 3】 前記指示取得手段は、前記ソースプログラムの中において、特定の配列データについて、グローバルメモリ領域に配置しない旨の指示を検出し、

前記最適化手段は、前記指示取得手段によって検出された指示の対象となった配列データについては、グローバルメモリ領域外のメモリ領域に配置する

ことを特徴とする請求項 1 記載のコンパイラ装置。

【請求項 4】 前記指示取得手段は、前記ソースプログラムの中において、特定の配列データについて、グローバルメモリ領域に配置する旨の指示を検出し、

前記最適化手段は、前記指示取得手段によって検出された指示の対象となった配列データについては、グローバルメモリ領域に配置する

ことを特徴とする請求項 1 記載のコンパイラ装置。

【請求項 5】 高級言語で記述されたソースプログラムが記録されたコンピュータ読み取り可能な記録媒体であって、

前記ソースプログラムには、当該ソースプログラムを機械語プログラムに翻訳するコンパイラに対して、（１）特定の配列データについて、グローバルメモリ領域に配置しない旨を指示する記述、及び、（２）特定の配列データについて、グローバルメモリ領域に配置する旨を指示する記述の少なくとも 1 つが含まれることを特徴とする記録媒体。

【請求項 6】 ソースプログラムを機械語プログラムに翻訳するコンパイラ装置であって、

生成する機械語プログラムを最適化する旨の指示を取得する指示取得手段と、取得された指示に従って機械語命令列を生成することにより、最適化を施す最適化手段とを備え、

前記最適化手段は、前記指示取得手段がソフトウェアパイプラインによる最適化についての指示を取得した場合には、取得した指示に従ってソフトウェアパイプラインに関する最適化を施す

ことを特徴とするコンパイラ装置。

【請求項 7】 前記指示取得手段は、前記ソースプログラムを翻訳する旨の指示とともに、ソフトウェアパイプラインによる最適化をしない旨の指示を取得し、

前記最適化手段は、前記指示取得手段がソフトウェアパイプラインによる最適化をしない旨の指示を取得した場合に、前記ソースプログラム中の全てのループ処理について、ソフトウェアパイプラインによる最適化が行われることを抑制する

ことを特徴とする請求項 6 記載のコンパイラ装置。

【請求項 8】 前記指示取得手段は、前記ソースプログラムの中において、特定のループ処理について、ソフトウェアパイプラインによる最適化をしな

い旨の指示を検出し、

前記最適化手段は、前記指示取得手段によって検出された指示の対象となったループ処理については、ソフトウェアパイプラインによる最適化が行われることを抑制する

ことを特徴とする請求項 6 記載のコンパイラ装置。

【請求項 9】 前記指示取得手段は、前記ソースプログラムの中において、特定のループ処理について、プロログ部及びエピログ部を除去したソフトウェアパイプラインによる最適化をする旨の指示を検出し、

前記最適化手段は、前記指示取得手段によって検出された指示の対象となったループ処理については、プロログ部及びエピログ部を除去できる範囲でソフトウェアパイプラインによる最適化をする

ことを特徴とする請求項 6 記載のコンパイラ装置。

【請求項 10】 前記指示取得手段は、前記ソースプログラムの中において、特定のループ処理について、プロログ部及びエピログ部を除去しないでソフトウェアパイプラインによる最適化をする旨の指示を検出し、

前記最適化手段は、前記指示取得手段によって検出された指示の対象となったループ処理については、プロログ部及びエピログ部を除去せずに、可能な範囲でソフトウェアパイプラインによる最適化をする

ことを特徴とする請求項 6 記載のコンパイラ装置。

【請求項 11】 前記指示取得手段は、前記ソースプログラムの中において、特定のループ処理の繰り返し回数に関する指定を検出し、

前記最適化手段は、前記指示取得手段によって検出された指示の対象となったループ処理については、指定された繰り返し回数に基づく最適化を行う

ことを特徴とする請求項 6 記載のコンパイラ装置。

【請求項 12】 前記繰り返し回数に関する指定は、前記ループ処理が繰り返される最低回数であり、

前記最適化手段は、前記最低回数がソフトウェアパイプラインによって重なり合うイタレーション数以上である場合に、ソフトウェアパイプラインによる最適化を行う

ことを特徴とする請求項 1 1 記載のコンパイラ装置。

【請求項 1 3】 高級言語で記述されたソースプログラムが記録されたコンピュータ読み取り可能な記録媒体であって、

前記ソースプログラムには、当該ソースプログラムを機械語プログラムに翻訳するコンパイラに対して、（１）特定のループ処理について、ソフトウェアパイプライニングによる最適化をしない旨を指示する記述、（２）特定のループ処理について、プロログ部及びエピログ部を除去したソフトウェアパイプライニングによる最適化をする旨を指示する記述、及び、（３）特定のループ処理について、プロログ部及びエピログ部を除去しないでソフトウェアパイプライニングによる最適化をする旨を指示する記述の少なくとも１つが含まれる

ことを特徴とする記録媒体。

【請求項 1 4】 ソースプログラムを機械語プログラムに翻訳するコンパイラ装置であって、

生成する機械語プログラムを最適化する旨の指示を取得する指示取得手段と、取得された指示に従って機械語命令列を生成することにより、最適化を施す最適化手段とを備え、

前記最適化手段は、前記指示取得手段がループアンローリングによる最適化についての指示を取得した場合には、取得した指示に従ってループアンローリングに関する最適化を施す

ことを特徴とするコンパイラ装置。

【請求項 1 5】 前記指示取得手段は、前記ソースプログラムを翻訳する旨の指示とともに、ループアンローリングによる最適化をしない旨の指示を取得し、

前記最適化手段は、前記指示取得手段がループアンローリングによる最適化をしない旨の指示を取得した場合に、前記ソースプログラム中の全てのループ処理について、ループアンローリングによる最適化が行われることを抑制する

ことを特徴とする請求項 1 4 記載のコンパイラ装置。

【請求項 1 6】 前記指示取得手段は、前記ソースプログラムの中において、特定のループ処理について、ループアンローリングによる最適化をする旨の指



示を検出し、

前記最適化手段は、前記指示取得手段によって検出された指示の対象となったループ処理については、ループアンローリングによる最適化を行う

ことを特徴とする請求項 1 4 記載のコンパイラ装置。

【請求項 1 7】 前記指示取得手段は、前記ソースプログラムの中において、特定のループ処理について、ループアンローリングによる最適化をしない旨の指示を検出し、

前記最適化手段は、前記指示取得手段によって検出された指示の対象となったループ処理については、ループアンローリングが行われることを抑制する

ことを特徴とする請求項 1 4 記載のコンパイラ装置。

【請求項 1 8】 前記指示取得手段は、前記ソースプログラムの中において、特定のループ処理の繰り返し回数に関する指定を検出し、

前記最適化手段は、前記指示取得手段によって検出された指定の対象となったループ処理については、指定された繰り返し回数に基づく最適化を行う

ことを特徴とする請求項 1 4 記載のコンパイラ装置。

【請求項 1 9】 前記繰り返し回数に関する指定は、前記ループ処理が繰り返される最低回数であり、

前記最適化手段は、前記最低回数が 1 以上である場合に、繰り返し回数が 0 である場合に必要とされるエスケープコードの生成を抑制する

ことを特徴とする請求項 1 8 記載のコンパイラ装置。

【請求項 2 0】 前記繰り返し回数に関する指定は、前記ループ処理が繰り返される最低回数であり、

前記最適化手段は、前記最低回数がループアンローリングによる展開数以上である場合に、ループアンローリングによる最適化を行う

ことを特徴とする請求項 1 8 記載のコンパイラ装置。

【請求項 2 1】 前記繰り返し回数に関する指定は、前記ループ処理が偶数回だけ繰り返される旨の保証であり、

前記最適化手段は、前記指示取得手段によって検出された指定の対象となったループ処理については、偶数回だけ繰り返されるものとして、ループアンローリ

ングによる最適化を行う

ことを特徴とする請求項 1 8 記載のコンパイラ装置。

【請求項 2 2】 前記繰り返し回数に関する指定は、前記ループ処理が奇数回だけ繰り返される旨の保証であり、

前記最適化手段は、前記指示取得手段によって検出された指定の対象となったループ処理については、奇数回だけ繰り返されるものとして、ループアンローリングによる最適化を行う

ことを特徴とする請求項 1 8 記載のコンパイラ装置。

【請求項 2 3】 高級言語で記述されたソースプログラムが記録されたコンピュータ読み取り可能な記録媒体であって、

前記ソースプログラムには、当該ソースプログラムを機械語プログラムに翻訳するコンパイラに対して、（１）特定のループ処理について、ループアンローリングによる最適化をする旨を指示する記述、（２）特定のループ処理について、ループアンローリングによる最適化をしない旨を指示する記述、及び、（３）特定のループ処理の繰り返し回数に関する保証を示す記述の少なくとも１つが含まれる

ことを特徴とする記録媒体。

【請求項 2 4】 ソースプログラムを機械語プログラムに翻訳するコンパイラ装置であって、

生成する機械語プログラムを最適化する旨の指示を取得する指示取得手段と、取得された指示に従って機械語命令列を生成することにより、最適化を施す最適化手段とを備え、

前記最適化手段は、前記指示取得手段が i f 変換についての指示を取得した場合には、取得した指示に従って i f 変換に関する最適化を施す

ことを特徴とするコンパイラ装置。

【請求項 2 5】 前記指示取得手段は、前記ソースプログラムを翻訳する旨の指示とともに、i f 変換をしない旨の指示を取得し、

前記最適化手段は、前記指示取得手段が i f 変換をしない旨の指示を取得した場合に、前記ソースプログラム中の全ての i f 構造文について、i f 変換が行わ

れることを抑制する

ことを特徴とする請求項 2 4 記載のコンパイラ装置。

【請求項 2 6】 前記指示取得手段は、前記ソースプログラムの中において、特定の i f 構造文について、i f 変換をする旨の指示を検出し、

前記最適化手段は、前記指示取得手段によって検出された指示の対象となった i f 構造文について、i f 変換をする

ことを特徴とする請求項 2 4 記載のコンパイラ装置。

【請求項 2 7】 前記指示取得手段は、前記ソースプログラムの中において、特定の i f 構造文について、i f 変換をしない旨の指示を検出し、

前記最適化手段は、前記指示取得手段によって検出された指示の対象となった i f 構造文については、i f 変換が行われることを抑制する

ことを特徴とする請求項 2 4 記載のコンパイラ装置。

【請求項 2 8】 高級言語で記述されたソースプログラムが記録されたコンピュータ読み取り可能な記録媒体であって、

前記ソースプログラムには、当該ソースプログラムを機械語プログラムに翻訳するコンパイラに対して、(1) 特定の i f 構造文について、i f 変換をする旨を指示する記述、及び、(2) 特定の i f 構造文について、i f 変換をしない旨を指示する記述の少なくとも 1 つが含まれる

ことを特徴とする記録媒体。

【請求項 2 9】 ソースプログラムを機械語プログラムに翻訳するコンパイラ装置であって、

生成する機械語プログラムを最適化する旨の指示を取得する指示取得手段と、取得された指示に従って機械語命令列を生成することにより、最適化を施す最適化手段とを備え、

前記最適化手段は、メモリ領域に配置する配列データのアライメントについての指示を取得した場合には、取得した指示に従ってデータをメモリ領域に配置することにより、前記最適化を施す

ことを特徴とするコンパイラ装置。

【請求項 3 0】 前記指示取得手段は、前記ソースプログラムを翻訳する旨

の指示とともに、特定の型の配列データのアライメントについての指定を取得し

前記最適化手段は、前記ソースプログラムで宣言された前記特定の型の全ての配列データについて、その先頭アドレスが前記アライメントとなるようにメモリ領域に配置する

ことを特徴とする請求項 2 9 記載のコンパイラ装置。

【請求項 3 1】 前記指示取得手段は、前記ソースプログラムを翻訳する旨の指示とともに、構造体のアライメントについての指定を取得し、

前記最適化手段は、前記ソースプログラムで宣言された前記構造体データについて、その先頭アドレスが前記アライメントとなるようにメモリ領域に配置することを特徴とする請求項 2 9 記載のコンパイラ装置。

【請求項 3 2】 前記指示取得手段は、前記ソースプログラムの中において、特定の変数名で示される引数のポインタ変数が指すデータのアライメントについての指定を検出し、

前記最適化手段は、前記指示取得手段によって検出された指定の対象となったデータについては、指定されたアライメントでメモリ領域に配置されているものとして、前記最適化を施す

ことを特徴とする請求項 2 9 記載のコンパイラ装置。

【請求項 3 3】 前記指示取得手段は、前記ソースプログラムの中において、特定の変数名で示されるローカルポインタ変数が指すデータのアライメントについての指定を検出し、

前記最適化手段は、前記指示取得手段によって検出された指定の対象となったデータについては、指定されたアライメントでメモリ領域に配置されているものとして、前記最適化を施す

ことを特徴とする請求項 2 9 記載のコンパイラ装置。

【請求項 3 4】 前記最適化手段は、メモリ領域に配置した前記データにアクセスするメモリアクセス命令については、2 以上のデータを同時に転送するペア命令を生成する

ことを特徴とする請求項 3 0 ～ 3 3 のいずれか 1 項に記載のコンパイラ装置。

【請求項 3 5】 高級言語で記述されたソースプログラムが記録されたコンピュータ読み取り可能な記録媒体であって、

前記ソースプログラムには、当該ソースプログラムを機械語プログラムに翻訳するコンパイラに対して、（１）特定の変数名で示される引数のポインタ変数が指すデータのアライメントを保証する旨の記述、及び、（２）特定の変数名で示されるローカルポインタ変数が指すデータのアライメントを保証する旨の記述の少なくとも１つが含まれる

ことを特徴とする記録媒体。

【請求項 3 6】 ソースプログラムを機械語プログラムに翻訳するコンパイラ装置のためのプログラムであって、

請求項 1 ～ 4、6 ～ 1 2、1 4 ～ 2 2、2 4 ～ 2 7、2 9 ～ 3 4 のいずれか 1 項に記載のコンパイラ装置が備える全ての手段としてコンピュータに機能させることを特徴とするプログラム。

【発明の詳細な説明】

【0 0 0 1】

【発明の属する技術分野】

本発明は、C 言語等の高級言語で記述されたソースプログラムを機械語プログラムに変換するコンパイラに関し、特に、コンパイラによる最適化についての指示に関する。

【0 0 0 2】

【従来の技術】

従来のコンパイラは、主に制御処理アプリケーション向けに作られている。制御処理アプリケーションでは、精密な実行性能及びコードサイズのチューニングはそれほど必要とされておらず、むしろ開発工数削減の観点から、ユーザは、「性能優先」、「コードサイズ優先」、あるいは、「性能とコードサイズのバランス」といった大雑把な指示（コンパイル時に指定するオプション等）のみ与えて、その最適化戦略のほとんどをコンパイラに任せている。

【0 0 0 3】

一方、クリティカルな実行性能及びコードサイズが要求されるメディア処理

アプリケーションの分野では、アセンブリ言語によるハンドコーディングを行うことにより、要求性能及びコードサイズを実現することを第一目標に開発が行われている。

## 【0004】

## 【発明が解決しようとする課題】

しかしながら、近年、メディア処理アプリケーションの増大化、多様化により、開発工数が増大し、メディア処理分野においても高級言語によるアプリケーション開発が必要とされるようになってきている。そのために、高級言語によるメディア処理アプリケーション開発を実現する試みが行われている。その際、ユーザは、高級言語開発であっても、より精密なチューニングができることを期待しており、コンパイラが行う最適化戦略を詳細に制御することが必要となる。

## 【0005】

したがって、従来のような大雑把な指示ではなく、コンパイラによる各種最適化の種類ごとにON/OFFやその程度を指定したり、プログラム中の変数やループ処理等の単位で最適化をON/OFFさせたりする等のきめ細かい制御が必要とされる。

## 【0006】

そこで、本発明は、このような状況に鑑みてなされたものであり、コンパイラによる最適化をユーザが緻密に制御することが可能な柔軟性の高いコンパイラを提供することを目的とする。

## 【0007】

## 【課題を解決するための手段】

上記目的を達成するために、本発明に係るコンパイラは、ソースプログラムを機械語プログラムに翻訳するコンパイラ装置であって、生成する機械語プログラムを最適化する旨の指示を取得する指示取得手段と、取得された指示に従って機械語命令列を生成することにより、最適化を施す最適化手段とを備え、前記最適化手段は、前記指示取得手段がグローバルメモリ領域に配置する配列データについての指示を取得した場合には、取得した指示に従ってグローバルメモリ領域に配置する配列データを決定することにより、前記最適化を施すことを特徴とする

【 0 0 0 8 】

また、本発明に係るコンパイラは、ソースプログラムを機械語プログラムに翻訳するコンパイラ装置であって、生成する機械語プログラムを最適化する旨の指示を取得する指示取得手段と、取得された指示に従って機械語命令列を生成することにより、最適化を施す最適化手段とを備え、前記最適化手段は、前記指示取得手段がソフトウェアパイプラインによる最適化についての指示を取得した場合には、取得した指示に従ってソフトウェアパイプラインに関する最適化を施すことを特徴とする。

【 0 0 0 9 】

また、本発明に係るコンパイラは、ソースプログラムを機械語プログラムに翻訳するコンパイラ装置であって、生成する機械語プログラムを最適化する旨の指示を取得する指示取得手段と、取得された指示に従って機械語命令列を生成することにより、最適化を施す最適化手段とを備え、前記最適化手段は、前記指示取得手段がループアンローリングによる最適化についての指示を取得した場合には、取得した指示に従ってループアンローリングに関する最適化を施すことを特徴とする。

【 0 0 1 0 】

また、本発明に係るコンパイラは、ソースプログラムを機械語プログラムに翻訳するコンパイラ装置であって、生成する機械語プログラムを最適化する旨の指示を取得する指示取得手段と、取得された指示に従って機械語命令列を生成することにより、最適化を施す最適化手段とを備え、前記最適化手段は、前記指示取得手段が i f 変換についての指示を取得した場合には、取得した指示に従って i f 変換に関する最適化を施すことを特徴とする。

【 0 0 1 1 】

また、本発明に係るコンパイラは、ソースプログラムを機械語プログラムに翻訳するコンパイラ装置であって、生成する機械語プログラムを最適化する旨の指示を取得する指示取得手段と、取得された指示に従って機械語命令列を生成することにより、最適化を施す最適化手段とを備え、前記最適化手段は、メモリ領域

に配置する配列データのアライメントについての指示を取得した場合には、取得した指示に従ってデータをメモリ領域に配置することにより、前記最適化を施すことを特徴とする。

【 0 0 1 2 】

なお、本発明は、上記のようなコンパイラ装置として実現することができるだけでなく、このようなコンパイラ装置が備える手段をステップとするプログラムとして実現したり、コンパイラへの指示が含まれたソースプログラムとして実現することもできる。そして、そのようなプログラムは、CD-ROM等の記録媒体やインターネット等の伝送媒体を介して広く流通させることができるのは言うまでもない。

【 0 0 1 3 】

【発明の実施の形態】

以下、本発明に係るコンパイラの実施の形態について図面を用いて詳細に説明する。

本実施の形態におけるコンパイラは、C言語等の高級言語で記述されたソースプログラムを特定のプロセッサが実行できる機械語プログラムに翻訳するクロスコンパイラであり、生成する機械語プログラムのコードサイズや実行時間に関する最適化の指示をきめ細かく指定することができるという特徴を有する。

【 0 0 1 4 】

まず、本実施の形態におけるコンパイラの対象となるプロセッサの一例について、図1～図36を用いて説明する。

【 0 0 1 5 】

本実施の形態におけるコンパイラの対象となるプロセッサは、例えば、通常のマイコンに比べて実行可能な命令の並列性が高く、AVメディア系信号処理技術分野をターゲットとして開発された汎用プロセッサである。

【 0 0 1 6 】

図1は、そのようなプロセッサの概略ブロック図の一例である。このプロセッサ1は、命令制御部10、デコード部20、レジスタファイル30、演算部40、I/F部50、命令メモリ部60、データメモリ部70、拡張レジスタ部80



及びI/Oインターフェース部90から構成される。演算部40は、SIMD型命令の演算を実行する算術論理・比較演算器41～43、乗算・積和演算器44、バレルシフタ45、除算器46及び変換器47からなる。乗算・積和演算器44は、ビット精度を落とさないように、最長で65ビットで累算する。また、乗算・積和演算器44は、算術論理・比較演算器41～43と同様、SIMD型命令の実行が可能である。更に、このプロセッサ1は、算術論理・比較演算命令が最大3並列実行可能である。

#### 【0017】

図2は、算術論理・比較演算器41～43の概略図を示す。算術論理・比較演算器41～43それぞれは、ALU部41a、飽和处理部41b及びフラグ部41cから構成される。ALU部41aは、算術演算器、論理演算器、比較器、TST器からなる。対応する演算データのビット幅は、8ビット(演算器を4並列で使用)、16ビット(演算器を2並列で使用)、32ビットである(全演算器で32ビットデータ処理)。更に算術演算結果に対しては、フラグ部41c等により、オーバーフローの検出とコンディションフラグの生成が行われる。各演算器、比較器、TST器の結果は、算術右シフト、飽和处理部41bによる飽和、最大・最小値検出、絶対値生成処理が行われる。

#### 【0018】

図3は、バレルシフタ45の構成を示すブロック図である。バレルシフタ45は、セクタ45a、45b、上位バレルシフタ45c、下位バレルシフタ45d及び飽和处理部45eから構成され、データの算術シフト(2の補数体系のシフト)または、論理シフト(符号なしシフト)を実行する。通常は、32ビットもしくは、64ビットのデータを入出力としている。レジスタ30a、30bに格納された被シフトデータに対して、別のレジスタまたは即値でシフト量が指定される。データは、左63ビット～右63ビットの算術または論理シフトが行われ、入力ビット長で出力される。

#### 【0019】

また、バレルシフタ45は、SIMD型命令に対して、8、16、32、64ビットのデータをシフトすることができる。例えば、8ビットデータのシフトを

4 並列で処理することができる。

#### 【0020】

算術シフトは、2の補数体系のシフトであり、加算や減算時の小数点の位置合わせや、2のべき乗の乗算(2、2の2乗、2の(-1)乗、2の(-2)乗倍など)等のために行われる。

#### 【0021】

図4は、変換器47の構成を示すブロック図である。変換器47は、飽和ブロック(SAT)47a、BSEQブロック47b、MSKGENブロック47c、VSUMBブロック47d、BCNTブロック47e及びILブロック47fから構成される。

#### 【0022】

飽和ブロック(SAT)47aは、入力データに対する飽和处理を行う。32ビットデータを飽和处理するブロックを2つ持つことにより、2並列のSIMD型命令をサポートする。

#### 【0023】

BSEQブロック47bは、MSBから連続する0か1をカウントする。

MSKGENブロック47cは、指定されたビット区間を1、それ以外を0として出力する。

VSUMBブロック47dは、入力データを指定されたビット幅に区切り、その総和を出力する。

BCNTブロック47eは、入力データで1となっているビットの数をカウントする。

ILブロック47fは、入力データを指定されたビット幅に区切り、各データブロックを入れ換えた値を出力する。

#### 【0024】

図5は、除算器46の構成を示すブロック図である。除算器46は、被除数を64ビット、除数を32ビットとし、商と剰余を32ビットずつ出力する。商と剰余を求めるまでに34サイクルを必要とする。符号付き、符号なし、両方のデータを扱うことが可能である。ただし、被除数と除数において符号の有無の設定は共通とする。その他、オーバーフローフラグ、0除算フラグを出力する機能を

有する。

#### 【 0 0 2 5 】

図 6 は、乗算・積和演算器 4 4 の構成を示すブロック図である。乗算・積和演算器 4 4 は、2 つの 3 2 ビット乗算器 (MUL) 4 4 a、4 4 b、3 つの 6 4 ビット加算器 (Adder) 4 4 c ~ 4 4 e、セレクトア 4 4 f 及び飽和处理部 (Saturation) 4 4 g から構成され、以下の乗算、積和演算を行う。

- ・ 3 2 × 3 2 ビットの signed の乗算、積和、積差演算
- ・ 3 2 × 3 2 ビットの unsigned の乗算
- ・ 1 6 × 1 6 ビットの 2 並列の signed の乗算、積和、積差演算
- ・ 3 2 × 1 6 ビットの 2 並列の signed の乗算、積和、積差演算

#### 【 0 0 2 6 】

これらの演算を整数、固定小数点フォーマット (h 1、h 2、w 1、w 2) のデータに対して行う。また、これらの演算に対し、丸め、飽和を行う。

#### 【 0 0 2 7 】

図 7 は、命令制御部 1 0 の構成を示すブロック図である。命令制御部 1 0 は、命令キャッシュ 1 0 a、アドレス管理部 1 0 b、命令バッファ 1 0 c ~ 1 0 e、ジャンプバッファ 1 0 f 及びローテーション部 (rotation) 1 0 g から構成され、通常時及び分岐時の命令供給を行う。1 2 8 ビットの命令バッファを 3 つ (命令バッファ 1 0 c ~ 1 0 e) 持つことにより、最大並列実行数に対応している。分岐処理に関しては、分岐実行前に、ジャンプバッファ 1 0 f 等を介して、後述する T A R レジスタに予め分岐先アドレスを格納する (settar 命令)。T A R レジスタに格納された分岐先アドレスを使用して、分岐を行う。

#### 【 0 0 2 8 】

なお、本実施の形態におけるコンパイラの対象となるプロセッサ 1 は、例えば、V L I W アーキテクチャを持つプロセッサである。ここで、V L I W アーキテクチャとは、1 つの命令語中に複数の命令 (ロード、ストア、演算、分岐など) を格納し、それらを全て同時に実行するアーキテクチャである。プログラマは、並列実行可能な命令を 1 つの発行グループとして記述することによって、その発行グループを並列処理させることができる。本明細書では、発行グループの区切り

を”;;”で示す。以下に表記例を示す。

(例 1)

```
mov r1, 0x23;;
```

この命令記述は、命令movのみを実行することを意味する。

(例 2)

```
mov r1, 0x38
```

```
add r0, r1, r2
```

```
sub r3, r1, r2;;
```

これらの命令記述は、命令mov、add、subを3並列で実行することを意味する。

【 0 0 2 9 】

命令制御部10は、発行グループを識別し、デコード部20に送る。デコード部20では、発行グループの命令を解析し、必要な資源を制御する。

【 0 0 3 0 】

次に、このようなプロセッサ1が備えるレジスタについて説明する。

プロセッサ1のレジスタセットは、以下の表1に示される通りである。

【表 1】

レジスタ名	ビット幅	本数	用途
R0～R31	32ビット	32本	汎用レジスタ。データメモリのポインタ、演算命令におけるデータの格納等に使用します。
TAR	32ビット	1本	分岐用レジスタ。分岐時の分岐アドレスの格納に使用します。
LR	32ビット	1本	リンク用レジスタ。
SVR	16ビット	2本	退避用レジスタ。コンディションフラグ(CFR)と各種モードを退避します。
M0～M1 (MH0:ML0～MH1:ML1)	64ビット	2本	演算用レジスタ。演算命令におけるデータの格納に使用します。

【 0 0 3 1 】

また、このようなプロセッサ1のフラグセット（後述する条件フラグレジスタ等で管理されるフラグ）は、以下の表2に示される通りである。

【表 2】

C0~C7	1ビット	8本	条件フラグ。条件成立・不成立を表します。
VC0~VC3	1ビット	4本	メディア処理拡張命令用条件フラグ。条件成立・不成立を表します。
OVS	1ビット	1本	オーバーフローフラグ。演算時のオーバーフローを検出します。
CAS	1ビット	1本	キャリーフラグ。演算時のキャリーを検出します。
BPO	5ビット	1本	ビット位置指定。マスク処理命令時に処理対象となるビット位置を指定します。
ALN	2ビット	1本	バイトアライン指定。
FXP	1ビット	1本	固定小数点演算モード。
UDR	32ビット	1本	未定義レジスタ

## 【0032】

図8は、汎用レジスタ（R0～R31）30aの構造を示す図である。汎用レジスタ（R0～R31）30aは、実行対象となっているタスクのコンテキストの一部を構成し、データまたはアドレスを格納する32ビットのレジスタ群である。なお、汎用レジスタR30およびR31は、それぞれグローバルポインタ、スタックポインタとして、ハードウェアが使用する。

## 【0033】

図9は、リンクレジスタ（LR）30cの構造を示す図である。なお、このリンクレジスタ（LR）30cと関連して、このプロセッサ1は、図示されていない退避レジスタ（SVR）も備える。リンクレジスタ（LR）30cは、関数コール時のリターンアドレスを格納する32ビットのレジスタである。なお、退避レジスタ（SVR）は、関数コール時の条件フラグレジスタのコンディションフラグ（CFR.CF）を退避する16ビットのレジスタである。リンクレジスタ（LR）30cは、後述する分岐レジスタ（TAR）と同様に、ループ高速化にも使用される。下位1ビットは常に0が読み出されるが、書き込み時には0を書き込む必要がある。

## 【0034】

例えば、call(brl,jmpl)命令を実行した場合には、このプロセッサ1は、リンクレジスタ（LR）30cに戻りアドレスを退避し、退避レジスタ（SVR）にコンディションフラグ（CFR.CF）を退避する。また、jmp命令を実行した場合には、リンクレジスタ（LR）30cから戻りアドレス（分岐先アドレス）を取り出し

、プログラムカウンタ（PC）を復帰させる。さらに、ret(jmpr)命令を実行した場合には、リンクレジスタ（LR）30cから分岐先アドレス(戻りアドレス)を取り出し、プログラムカウンタ（PC）に格納(復帰)する。さらに、退避レジスタ（SVR）からコンディションフラグを取り出し、条件フラグレジスタ（CFR）32のコンディションフラグ領域CFR.CFに格納(復帰)する。

## 【0035】

図10は、分岐レジスタ（TAR）30dの構造を示す図である。分岐レジスタ（TAR）30dは、分岐ターゲットアドレスを格納する32ビットのレジスタである。主に、ループの高速化に用いられる。下位1ビットは常に0が読み出されるが、書き込み時には0を書き込む必要がある。

## 【0036】

例えば、jmp,jloop命令を実行した場合には、プロセッサ1は、分岐レジスタ（TAR）30dから分岐先アドレスを取り出し、プログラムカウンタ（PC）に格納する。分岐レジスタ（TAR）30dに格納されたアドレスの命令が分岐用命令バッファに格納されている場合は、分岐ペナルティが0になる。分岐レジスタ（TAR）30dにループの先頭アドレスを格納しておくことでループを高速化することができる。

## 【0037】

図11は、プログラム状態レジスタ（PSR）31の構造を示す図である。プログラム状態レジスタ（PSR）31は、実行対象となっているタスクのコンテキストの一部を構成し、以下に示されるプロセッサ状態情報を格納する32ビットのレジスタである。

## 【0038】

ビットSWE：VMP（Virtual Multi-Processor）のLP（Logical Processor）切替えイネーブルを示す。「0」はLP切替え不許可を示し、「1」はLP切替え許可を示す。

## 【0039】

ビットFXP：固定小数点モードを示す。「0」はモード0を示し、「1」はモード1を示す。

ビットIH：割込み処理フラグであり、マスカブル割込み処理中であることを示す。「1」は割込み処理中であることを示し、「0」は割込み処理中でないことを示す。割込みが発生すると自動的にセットされる。rti命令で割込みから復帰したところが、他の割込み処理中かプログラム処理中であるのかを見分けるために使用される。

## 【0040】

ビットEH：エラーまたはNMIを処理中であることを示すフラグである。「0」はエラー/NMI割込み処理中でないことを示し、「1」はエラー/NMI割込み処理中であることを示す。EH=1のとき、非同期エラーまたはNMIが発生した場合は、マスクされる。また、VMPイネーブル時はVMPのプレート切り替えがマスクされる。

## 【0041】

ビットPL[1:0]：特権レベルを示す。「00」は特権レベル0、つまり、プロセッサアブストラクションレベルを示し、「01」は特権レベル1（設定できない）を示し、「10」は特権レベル2、つまり、システムプログラムレベルを示し、「11」は特権レベル3、つまり、ユーザプログラムレベルを示す。

## 【0042】

ビットLPIE3：LP固有割込み3イネーブルを示す。「1」は割込み許可を示し、「0」は割込み不許可を示す。

ビットLPIE2：LP固有割込み2イネーブルを示す。「1」は割込み許可を示し、「0」は割込み不許可を示す。

ビットLPIE1：LP固有割込み1イネーブルを示す。「1」は割込み許可を示し、「0」は割込み不許可を示す。

ビットLPIE0：LP固有割込み0イネーブルを示す。「1」は割込み許可を示し、「0」は割込み不許可を示す。

ビットAEE：ミスアライメント例外イネーブルを示す。「1」はミスアライメント例外許可を示し、「0」はミスアライメント例外不許可を示す。

ビットIE：レベル割込みイネーブルを示す。「1」はレベル割込み許可を示し、「0」はレベル割込み不許可を示す。

ビットIM[7:0] : 割込みマスクを示す。レベル0～7まで定義され、個々のレベルでマスクすることができる。レベル0が最も高いレベルとなる。IMによりマスクされていない割込み要求のうち最も高いレベルを持った割込み要求のみがプロセッサ1に受理される。割込み要求を受理すると受理したレベル以下のレベルはハードウェアで自動的にマスクされる。IM[0]はレベル0のマスクであり、IM[1]はレベル1のマスクであり、IM[2]はレベル2のマスクであり、IM[3]はレベル3のマスクであり、IM[4]はレベル4のマスクであり、IM[5]はレベル5のマスクであり、IM[6]はレベル6のマスクであり、IM[7]はレベル7のマスクである。

reserved : 予約ビットを示す。常に0が読み出される。書き込む時は0を書き込む必要がある。

#### 【0043】

図12は、条件フラグレジスタ(CFR)32の構造を示す図である。条件フラグレジスタ(CFR)32は、実行対象となっているタスクのコンテキストの一部を構成する32ビットのレジスタであり、コンディションフラグ(条件フラグ)、オペレーションフラグ(演算フラグ)、ベクタコンディションフラグ(ベクタ条件フラグ)、演算命令用ビット位置指定フィールド、SIMDデータアライン情報フィールドから構成される。

#### 【0044】

ビットALN[1:0] : アラインモードを示す。valnvc命令のアラインモードを設定する。

ビットBPO[4:0] : ビットポジションを示す。ビット位置指定の必要な命令で使用する。

ビットVC0～VC3 : ベクタ条件フラグである。LSB側のバイトあるいはハーフワードから順にVC0に対応し、MSB側がVC3に対応する。

ビットOVS : オーバーフローフラグ(サマリー)である。飽和発生やオーバーフロー検出でセットされる。検出されなかった場合は、命令実行前の値を保持する。クリアはソフトで行う必要がある。

ビットCAS : キャリーフラグ(サマリー)である。addc命令でキャリーまたはsubc命令でボローが発生した場合セットされる。addc命令でキャリーもしくはsu



bc命令でボローが発生しなかった場合は、命令実行前の値を保持する。クリアはソフトで行う必要がある。

ビットC0～C7：コンディションフラグである。条件付き実行命令における条件(TRUE/FALSE)を示す。条件付き命令の条件とビットC0～C7との対応は、命令に含まれるプレディケート・ビットによって決定される。なお、フラグC7は常に値が1である。フラグC7へのFALSE条件の反映(0書き込み)は無視される。

reserved：予約ビットを示す。常に0が読み出される。書き込む時は0を書き込む必要がある。

#### 【0045】

図13は、アキュムレータ(M0, M1)30bの構造を示す図である。このアキュムレータ(M0, M1)30bは、実行対象となっているタスクのコンテキストの一部を構成し、図13(a)に示される32ビットレジスタMH0-MH1(乗除算・積和用レジスタ(上位32ビット))と、図13(b)に示される32ビットレジスタML0-ML1乗除算・積和用レジスタ(下位32ビット)とからなる。

#### 【0046】

レジスタMH0-MHは、乗算命令では結果の上位32ビットを格納するのに使用される。積和命令ではアキュムレータの上位32ビットとして使用される。また、ビットストリームを取り扱う場合に汎用レジスタと組み合わせて使用することができる。レジスタML0-ML1は、乗算命令では結果の下位32ビットを格納するのに使用される。積和命令ではアキュムレータの下位32ビットとして使用される。

#### 【0047】

図14は、プログラムカウンタ(PC)33の構造を示す図である。このプログラムカウンタ(PC)33は、実行対象となっているタスクのコンテキストの一部を構成し、実行中の命令のアドレスを保持する32ビットのカウンタである。下位1ビットは常に0が格納される。

#### 【0048】

図15は、PC退避用レジスタ(IPC)34の構造を示す図である。このP

C退避用レジスタ（IPC）34は、実行対象となっているタスクのコンテキストの一部を構成する32ビットのレジスタであり、下位1ビットは常に0が読み出されるが、書き込み時には0を書き込む必要がある。

【0049】

図16は、PSR退避用レジスタ（IPSR）35の構造を示す図である。このPSR退避用レジスタ（IPSR）35は、実行対象となっているタスクのコンテキストの一部を構成し、プログラム状態レジスタ（PSR）31を退避するための32ビットのレジスタであり、プログラム状態レジスタ（PSR）31の予約ビットに対応する部分は常に0が読み出されるが、書き込み時には0を書き込む必要がある。

【0050】

次に、本実施の形態におけるコンパイラの対象となるプロセッサ1のメモリ空間について説明する。例えば、プロセッサ1では、4GBのリニアなメモリ空間を32分割し、128MB単位の空間に命令SRAM（Static RAM）とデータSRAMが割り当てられる。この128MBの空間を1ブロックとして、SAR（SRAM Area Register）にアクセスしたいブロックを設定する。アクセスされたアドレスがSARで設定された空間である場合は、直接命令SRAM/データSRAMに対してアクセスを行うが、SARで設定された空間でない場合は、バスコントローラ（BCU）に対してアクセス要求を出す。BCUにはオン・チップ・メモリ（OCM）、外部メモリ、外部デバイス、I/Oポート等が接続されており、それらのデバイスに対して読み書きを行うことができる。

【0051】

図17は、本実施の形態におけるコンパイラの対象となるプロセッサ1のパイプライン動作を示すタイミング図である。このプロセッサ1は、本図に示されるように、例えば、基本的に命令フェッチ、命令割り当て（ディスパッチ）、デコード、実行、書き込みの5段パイプラインで構成されている。

【0052】

図18は、このようなプロセッサ1による命令実行時の各パイプライン動作を示すタイミング図である。命令フェッチステージでは、プログラムカウンタ（P

C) 33で指定されるアドレスの命令メモリをアクセスし、命令を命令バッファ 10c~10e等に転送する。命令割り当てステージでは、分岐系命令に対する分岐先アドレス情報の出力、入力レジスタ制御信号の出力、可変長命令の割り当てを行い、命令をインストラクションレジスタ（IR）に転送する。デコードステージでは、IRをデコード部20に入力し、演算器制御信号、メモリアクセス信号を出力する。実行ステージでは、演算を実行、演算結果をデータメモリか汎用レジスタ（R0~R31）30aに出力する。書き込みステージでは、データ転送、演算結果を汎用レジスタに格納する。

#### 【0053】

本実施の形態におけるコンパイラの対象となるプロセッサ1は、例えば、VLIWアーキテクチャにより上記の処理を最高3並列で行うことができる。したがって、図18に示された動作については、本プロセッサ1は、図19に示されるタイミングで並列に実行する。

#### 【0054】

次に、以上のように構成されたプロセッサ1の命令セットの例について説明する。

#### 【0055】

以下の表3~表5は、本実施の形態におけるコンパイラの対象となるプロセッサ1が実行する命令をカテゴリー別に分類した表である。

【表 3】

カテゴリー	演算器	命令オペコード
メモリ転送命令(ロード)	M	ld, ldh, ldhu, ldb, ldbu, ldp, ldhp, ldbp, ldbh, ld buh, ldbhp, ldbuhp
メモリ転送命令(ストア)	M	st, sth, stb, stp, sthp, stbp, stbh, stbhp
メモリ転送命令(その他)	M	dpref, ldstb
外部レジスタ転送命令	M	rd, rde, wt, wte
分岐命令	B	br, brl, call, jmp, jmp1, jmpr, ret, jmpf, jloop, s etbb, setlr, settar
ソフトウェア割込み命令	B	rti, pi0, pi0l, pi1, pi1l, pi2, pi2l, pi3, pi3l, pi 4, pi4l, pi5, pi5l, pi6, pi6l, pi7, pi7l, sc0, sc1, sc2, sc3, sc4, sc5, sc6, sc7
VMP/割込み制御命令	B	intd, inte, vmpsleap, vmpsus, vmpswd, vmpswe, vm pwait
算術演算命令	A	abs, absvh, absvw, add, addarvw, addc, addmsk, ad ds, addsr, addu, addvh, addvw, neg, negvh, negvw, rsub, sladd, s2add, sub, subc, submsk, subs, subv h, subvw, max, min
論理演算命令	A	and, andn, or, sethi, xor, not
比較命令	A	cmpCC, cmpCCa, cmpCCn, cmpCCo, tstn, tstna, tstn n, tstno, tstz, tstza, tstzn, tstzo
転送命令	A	mov, movcf, mvclcas, mvclovs, setfo, vcchk
NOP命令	A	nop
シフト命令1	S1	asl, aslvh, aslvw, asr, asrvh, asrvw, lsl, lsr, ro l, ror
シフト命令2	S2	aslp, aslpvw, asrp, asrpvw, lslp, lsrp

【表 4】

カテゴリー	演算器	命令オペコード
抽出命令	S2	ext, extb, extbu, exth, exthu, extr, extru, extu
マスク命令	C	msk, mskgen
飽和命令	C	satl2, sat9, satb, satbu, sath, satw
変換命令	C	valn, valn1, valn2, valn3, valnvc1, valnvc2, valnvc3, valnvc4, vhpkb, vhpkh, vhunpkb, vhunpkh, vintlhb, vintlhh, vintlhb, vintlhh, vlpkb, vlpkb u, vlpkh, vlpkhu, vlunpkb, vlunpkbu, vlunpkh, vlunpkhu, vstovb, vstovh, vunpk1, vunpk2, vxchngh, vexth
ビットカウント命令	C	bcnt1, bseq, bseq0, bseq1
その他	C	byterev, extw, mskbrvb, mskbrvh, rndvh, movp
乗算命令1	X1	fmulhh, fmulhhr, fmulhw, fmulhww, hmul, lmul
乗算命令2	X2	fmulww, mul, mulu
積和命令1	X1	fmachh, fmachhr, fmachw, fmachww, hmac, lmac
積和命令2	X2	fmacww, mac
積差命令1	X1	fmsuhh, fmsuhhr, fmsuhw, fmsuww, hmsu, lmsu
積差命令2	X2	fmsuww, msu
割算命令	DIV	div, divu
デバッガ命令	DBGM	dbgm0, dbgm1, dbgm2, dbgm3

【表 5】

カテゴリー	演算器	命令オペコード
SIMD 算術演算命令	A	vabshvh, vaddb, vaddh, vaddhvc, vaddhvh, vaddrh vc, vaddsb, vaddsh, vaddsrh, vaddsrh, vasubb, vc chk, vhaddh, vhaddhvh, vhsabh, vhsabhvh, vladdh vladdhvh, vlsubh, vlsubhvh, vnegb, vnegh, vneg hvh, vsaddb, vsaddh, vsgrh, vsrsubb, vsrsubh, vs subb, vssubh, vsubb, vsubh, vsubhvh, vsubsh, vsu mh, vsumh2, vsumrh2, vxaddh, vxaddhvh, vxsubh, v xsubhvh, vmaxb, vmaxh, vminb, vminh, vmovt, vse
SIMD 比較命令	A	vcmpqeb, vcmpqeh, vcmpgeb, vcmpgeh, vcmpgtb, vc mpgth, vcmpleb, vcmpleh, vcmpltb, vcmplth, vcmp neb, vcmpneh, vscmpqeb, vscmpqeh, vscmpgeb, vscmpgeh, vscmpg tb, vscmpgth, vscmpleb, vscmpleh, vscmpltb, vsc mplth, vscmpneb, vscmpneh
SIMD シフト命令1	S1	vaslb, vaslh, vaslvh, vasrb, vasrh, vasrvh, vlsl b, vlslh, vlslb, vlslh, vlslh, vlslh, vlslh, vlslh h
SIMD シフト命令2	S2	vasl, vaslvw, vasr, vasrvw, vlsl, vlslr
SIMD 飽和命令	C	vsath, vsath12, vsath8, vsath8u, vsath9
SIMD その他の命令	C	vabssumb, vrndvh
SIMD 乗算命令	X2	vfmulh, vfmulhr, vfmulw, vhfmulh, vhfmulhr, vhf mulw, vhmul, vlfmulh, vlfmulhr, vlfmulw, vlmul, vmul, vpfmulhww, vxfmulh, vxfmulhr, vxfmulw, vx mul
SIMD 積和命令	X2	vfmach, vfmachr, vfmacw, vhfmach, vhfmacr, vhf macw, vhmach, vlfmach, vlfmacr, vlfmacw, vlmach, vmach, vpfmachww, vxfmach, vxfmachr, vxfmachw, vx mach
SIMD 積差命令	X2	vfmsuh, vfmsuw, vhfmsuh, vhfmsuw, vhfmsu, vlfmsu h, vlfmsuw, vlmsu, vmsu, vxfmsuh, vxfmsuw, vxmsu

【0056】

なお、表中の「演算器」は、その命令が使用する演算器を示す。演算器の略号の意味は次の通りである。つまり、「A」はALU命令、「B」は分岐命令、「C」は変換命令、「DIV」は除算命令、「DBGM」はデバッグ命令、「M」はメモリアクセス命令、「S1」、「S2」はシフト命令、「X1」、「X2」は乗算命令を意味する。

【0057】

図20は、このようなプロセッサ1が実行する命令のフォーマットの例を示す図である。そのフォーマットには、図20(a)に示される16ビット命令フォ

フォーマットと、図20(b)に示される32ビット命令フォーマットとがある。

【0058】

なお、図中における略号の意味は次の通りである。つまり、「E」はエンドビット（並列実行の境界）、「F」はフォーマットビット（00、01、10：16ビット命令フォーマット、11：32ビット命令フォーマット）、「P」はプレディケート（実行条件：8個の条件フラグC0～C7のいずれかを指定）、「OP」はオペコードフィールド、「R」はレジスタフィールド、「I」は即値フィールド、「D」ディスプレイメントフィールドを意味する。なお、「E」フィールドはVLIWに特有のもので、E=0の命令は次の命令と並列に実行される。つまり、「E」フィールドによって並列度が可変のVLIWを実現している。

【0059】

図21～図36は、プロセッサ1が実行する命令の概略的な機能を説明する図である。つまり、図21は、カテゴリー「ALUadd（加算）系」に属する命令を説明する図であり、図22は、カテゴリー「ALUsub（減算）系」に属する命令を説明する図であり、図23は、カテゴリー「ALUlogic（論理演算）系ほか」に属する命令を説明する図であり、図24は、カテゴリー「CMP（比較演算）系」に属する命令を説明する図であり、図25は、カテゴリー「mul（乗算）系」に属する命令を説明する図であり、図26は、カテゴリー「mac（積和演算）系」に属する命令を説明する図であり、図27は、カテゴリー「msu（積差演算）系」に属する命令を説明する図であり、図28は、カテゴリー「MEMld（メモリ読み出し）系」に属する命令を説明する図であり、図29は、カテゴリー「MEMstore（メモリ書き出し）系」に属する命令を説明する図であり、図30は、カテゴリー「BRA（分岐）系」に属する命令を説明する図であり、図31は、カテゴリー「BSasl（算術バレルシフト）系ほか」に属する命令を説明する図であり、図32は、カテゴリー「BSlsr（論理バレルシフト）系ほか」に属する命令を説明する図であり、図33は、カテゴリー「CNVvaln（算術変換）系」に属する命令を説明する図であり、図34は、カテゴリー「CNV（一般変換）系」に属する命令を説明する図であり、図35は、カテゴリー「SATvlpk（飽和处理）系」に属する命令を説明する図であり、図36は、カテゴリー「ETC（その他）系」に属する命

令を説明する図である。

# 【0060】

これらの図において、項目「SIMD」は、その命令の型（SISD（SINGLE）かSIMDかの区別）を示し、項目「サイズ」は、演算の対象となる個々のオペランドのサイズを示し、項目「命令」は、その命令のオペコードを示し、項目「オペランド」は、その命令のオペランドを示し、項目「CFR」は、条件フラグレジスタの変化を示し、項目「PSR」は、プロセッサ状態レジスタの変化を示し、項目「代表的な動作」は、動作の概要を示し、項目「演算器」は、使用される演算器を示し、項目「3116」は、命令のサイズを示す。

# 【0061】

以下に、後述する具体例で使用される主な命令についてのプロセッサ1の動作を説明する。

ld Rb, (Ra, D10)

レジスタRaにディスプレースメント値(D10)を加算したアドレスからワードデータをレジスタRbにロードする。

ldh Rb, (Ra+) I9

レジスタRaが示すアドレスからハーフワードデータを符号拡張してロードする。さらに、レジスタRaに即値(I9)を加算し、レジスタRaに格納する。

ldp Rb:Rb+1, (Ra+)

レジスタRaが示すアドレスから レジスタRbとRb+1に2つのワードデータを符号拡張してロードする。さらに、レジスタRaに8を加算し、レジスタRaに格納する。

# 【0062】

ldhp Rb:Rb+1, (Ra+)

レジスタRaが示すアドレスから2つのハーフワードデータを符号拡張してロードする。さらに、レジスタRaに4を加算し、レジスタRaに格納する。

setlo Ra, I16

レジスタRaに即値(I16)を符号拡張して格納する。

sethi Ra, I16



レジスタRaの上位 16 bitに即値(I16)を格納する。レジスタRaの下位 16 bitには影響しない。

【0063】

ld Rb, (Ra)

レジスタRaが示すアドレスからワードデータをレジスタRbにロードする。

add Rc, Ra, Rb

レジスタRaとRbを加算し、レジスタRbに格納する。

addu Rb, GP, I13

レジスタGPに即値(I13)を加算し、レジスタRbに格納する。

st (GP, D13), Rb

レジスタGPにディスプレースメント値(D13)を加算したアドレスに、レジスタRbに格納されたハーフワードデータをストアする。

【0064】

sth (Ra+) I9, Rb

レジスタRaが示すアドレスに、レジスタRbに格納されたハーフワードデータをストアする。さらに、レジスタRaに即値(I9)を加算し、レジスタRaに格納する。

stp (Ra+), Rb:Rb+1

レジスタRaが示すアドレスに、レジスタRbとRb+1に格納された 2 つのワードデータをストアする。さらに、レジスタRaに 8 を加算し、レジスタRaに格納する。

ret

サブルーチンコールからのリターンに使用する。LRに格納されているアドレスに分岐する。SVR.CFをCFR.CFに転送する。

【0065】

mov Ra, I16

レジスタRaに値(I16)を符号拡張して格納する。

settar C6, D9

以下の処理を行う。(1) PCとディスプレースメント値(D9)を加算したアドレスを 分岐レジスタTARに格納する。(2) そのアドレスの命令をフェッチして分

岐用命令バッファに格納する。(3) C6を 1 にセットする。

settar C6,Cm,D9

以下の処理を行う。(1) PCとディスプレースメント値(D9)を加算したアドレスを 分岐レジスタTARに格納する。(2) そのアドレスの命令をフェッチして分岐用命令バッファに格納する。(3) C6を 1 に、Cmを 0 にセットする。

【 0 0 6 6 】

settar C6,C2:C4,D9

以下の処理を行う。(1) PCとディスプレースメント値(D9)を加算したアドレスを 分岐レジスタTARに格納する。(2) そのアドレスの命令をフェッチして分岐用命令バッファに格納する。(3) C4とC6を 1 に、C2とC3を 0 にセットする。

jloop C6,TAR,Ra2,-1

ループで使用する。以下の処理を行う。(1) レジスタRa2に -1 を加算し、レジスタRa2に格納する。レジスタRa2が 0 より小さくなるとC6に 0 をセットする。(2) 分岐レジスタTARが示すアドレスにジャンプする。

jloop C6,Cm,TAR,Ra2,-1

ループで使用する。以下の処理を行う。(1) Cm に 1 をセットする。(2) レジスタRa2に -1 を加算し、レジスタRa2に格納する。レジスタRa2が 0 より小さくなるとC6に 0 をセットする。(3) 分岐レジスタTARが示すアドレスにジャンプする。

【 0 0 6 7 】

jloop C6,C2:C4,TAR,Ra2,-1

ループで使用する。以下の処理を行う。(1) C3をC2に転送し、C4をC3とC6に転送する。(2) レジスタRa2に -1 を加算し、レジスタRa2に格納する。レジスタRa2が 0 より小さくなるとC4に 0 をセットする。(3) 分岐レジスタTARが示すアドレスにジャンプする。

mul Mm,Rb,Ra,I8

レジスタRaと即値(I8)を符号付乗算し、結果をレジスタMmとレジスタRbに格納する。

mac Mm,Rc,Ra,Rb,Mn

レジスタRaとRbを整数乗算し、レジスタMnと加算する。結果をレジスタMmとレジスタRcに格納する。

【0068】

lmac Mm,Rc,Ra,Rb,Mn

レジスタRbをハーフワードベクタ形式で扱う。レジスタRaとRbの下位 16 bit を整数乗算し、レジスタMnと加算する。結果をレジスタMmとレジスタRcに格納する。

jloop C6,C2:C4,TAR,Ra2,-1

ループで使用する。以下の処理を行う。(1) C3をC2に転送し、C4をC3とC6に転送する。(2) レジスタRa2に -1 を加算し、レジスタRa2に格納する。レジスタRa2が 0 より小さくなるとC4に 0 をセットする。(3) 分岐レジスタTARが示すアドレスにジャンプする。

asr Rc,Ra,Rb

レジスタRaをRbが示すビット数だけ算術右シフトする。レジスタRbは±31 以内に飽和され、負の場合は算術左シフトになる。

br D9

現在のPCに、ディスプレースメント値(D9)を加算し、そのアドレスに分岐する。

【0069】

jmpf TAR

分岐レジスタTARに格納されているアドレスに分岐する。

cmpCC Cm,Ra,I5

CC には次のCC比較条件を記述可能である。

eq/ne/gt/ge/gtu/geu/le/lt/leu/ltu

CCがeq/ne/gt/ge/le/ltの場合、I5は符号付きの値で、符号拡張して比較する。  
。CCがgtu/geu/leu/ltuの場合、I5は符号なしの値である。

【0070】

[コンパイラ]

次に、以上のプロセッサ1をターゲットとする本実施の形態におけるコンパイラについて説明する。

#### 【0071】

図37は、本実施の形態におけるコンパイラ100の構成を示す機能ブロック図である。このコンパイラ100は、C言語等の高級言語で記述指定されたソースプログラム101を、上述のプロセッサ1をターゲットプロセッサとする機械語プログラム102に変換するクロスコンパイラであり、パーソナルコンピュータ等のコンピュータ上で実行されるプログラムによって実現され、大きく分けて、解析部110と、最適化部120と、出力部130とから構成される。

#### 【0072】

解析部110は、コンパイルの対象となるソースプログラム101及びこのコンパイラ100に対するユーザからの指示等を字句解析することによって、コンパイラ100に対する指示（オプション及びプラグマ）については最適化部120や出力部130に伝達し、コンパイルの対象となるプログラムについては内部形式データに変換したりする。

#### 【0073】

なお、「オプション」とは、コンパイラ100を起動する際に、コンパイルの対象となるソースプログラム101の指定とともに、ユーザが任意に指定することができるコンパイラ100への指示であり、生成する機械語プログラム102のコードサイズ及び実行時間を最適化するための指示等が含まれる。例えば、ユーザは、ソースプログラム101「sample.c」をコンパイルするときに、コマンド「ammp-cc」を用いて、コンピュータ上で、

```
c:¥>ammp-cc -o -max-gp-datasize=40 sample.c
```

と入力することができる。このコマンドにおける付加的な指示「-o」及び「-max-gp-datasize=40」がオプションである。このようなオプションによる指示は、ソースプログラム101全体に対する指示として扱われる。

#### 【0074】

また、「プラグマ（又は、プラグマ指令）」とは、ソースプログラム101中にユーザが任意に指定（配置）することができるコンパイラ100への指示であ

り、オプションと同様に、生成する機械語プログラム102のコードサイズ及び実行時間を最適化するための指示等が含まれる。本実施の形態におけるコンパイラ100では、「#pragma」で始まる文字列である。例えば、ユーザは、ソースプログラム101中に、

【0075】

```
#pragma_no_gp_access 変数名
```

というステートメントを記述しておくことができる。このステートメントがプラグマ（プラグマ指令）である。このようなプラグマは、オプションと異なり、当該プラグマの直後に配置された変数やループ処理等だけに対する個別的な指示として扱われる。

【0076】

最適化部120は、解析部110から出力されたソースプログラム101（内部形式データ）に対して、解析部110からの指示等に従って、（1）実行速度の向上を優先した最適化、（2）コードサイズの削減を優先した最適化、（3）実行速度とコードサイズの両方の最適化、の中から選択された最適化を実現するための全体的な最適化処理を行うことに加えて、ユーザによるオプション及びプラグマによって指定された個別的な最適化処理を行う処理部（グローバル領域割り付け部121、ソフトウェアパイプライン部122、ループアンローリング部123、if変換部124及びペア命令生成部125）を有する。

【0077】

グローバル領域割り付け部121は、グローバル領域（共通のデータ領域として関数を超えて参照可能なメモリ領域）に配置する変数（配列）の最大データサイズの指定、グローバル領域に配置させる変数の指定、及び、グローバル領域に配置させない変数の指定に関するオプション及びプラグマに従った最適化処理を行う。

【0078】

ソフトウェアパイプライン部122は、ソフトウェアパイプラインを行わない旨の指示、プロログ部・エピログ部が除去できる範囲でソフトウェアパイプラインを行う旨の指示、及び、プロログ部・エピログ部を除去せずに可

能な範囲でソフトウェアパイプラインを行う旨の指示に関するオプション及びプラグマに従った最適化処理を行う。

## 【0079】

ループアンローリング部123は、ループアンローリングを行う旨の指示、ループアンローリングを行わない旨の指示、ループが繰り返される最低回数の保証、ループが偶数回繰り返される旨の保証、及び、ループが奇数回繰り返される旨の保証に関するオプション及びプラグマに従った最適化処理を行う。

## 【0080】

if変換部124は、if変換を行う旨の指示、及び、if変換を行わない旨の指示に関するオプション及びプラグマに従った最適化処理を行う。

## 【0081】

ペア命令生成部125は、配列と構造体の先頭アドレスのアラインの指定、及び、関数引数のポインタ変数やローカルポインタ変数の指すデータのアライメントの保証に関するプラグマに従った最適化処理を行う。

## 【0082】

出力部130は、最適化部120による最適化処理が施されたソースプログラム101に対して、内部形式データを対応する機械語命令に置き換えたり、ラベルやモジュール等のアドレスを解決したりすることで、機械語プログラム102を生成し、ファイル等として出力する。

## 【0083】

次に、以上のように構成された本実施の形態におけるコンパイラ100の特徴的な動作について具体例を示しながら説明する。

## [グローバル領域割り付け部121]

まず、グローバル領域割り付け部121の動作とその意義について説明する。グローバル領域割り付け部121は、大きく分けて、(1)グローバル領域配置の最大データサイズの指定に関する最適化と、(2)グローバル領域配置の指定に関する最適化とを行う。

## 【0084】

まず、(1)グローバル領域配置の最大データサイズの指定に関する最適化に

ついて説明する。

上記プロセッサ1には、グローバルポインタレジスタ (gp; 汎用レジスタ R30) が用意されており、グローバル領域 (以下 gp 領域とする) の先頭のアドレスを保持している。gp 領域先頭からのディスプレースメントが最大14ビットの範囲については、1命令でアクセスすることが可能である。

【0085】

この gp 領域には、外部変数・静的変数等の配列を配置することが可能である。ただし、1命令でアクセスできる範囲を超えた場合、逆に性能が低下するため注意が必要である。

【0086】

図38(a)は、グローバル領域におけるデータ等の配置例を示す図である。ここでは、配列Aのデータサイズは、最大データサイズを超えない値であり、配列Cのデータサイズは、最大データサイズを超える値である。

【0087】

gp 領域に実体が収まっている配列Aへのアクセスは、以下の例のように、1命令で可能である。

例: `ld r1, (gp, _A - .MN.gptop);;`

なお、この例において、「.MN.gptop」は、グローバルポインタレジスタと同じアドレスを指すセクション名 (ラベル) である。

【0088】

一方、gp 領域配置の最大データサイズを超える配列Cの場合は、gp 領域以外に実体が配置され、配列Cのアドレスのみが gp 領域に配置される (なお、後述の、`#pragma _no_gp_access`指令を使用した場合は、gp 領域に実体もアドレスも格納されない)。

【0089】

この場合、配列Cへのアクセスは、以下の例のように、複数命令必要になる。

例: gpアドレス間接アクセスの場合

`ld r1, (gp, _C$ - .MN.gptop);;`

`ld r1, (r1, 8);;`

例：絶対アドレスアクセスの場合

```
setlo r0,L0(_C+8);;
```

```
sethi r0,HI(_C+8);;
```

```
ld    r0,(r0);;
```

【0090】

なお、図38(b)に示されるグローバル領域以外の領域における配置例のように、gp領域の1命令でアクセスできる範囲外に実体が配置された配列Zの場合でも、以下のようなコードが生成される。

```
ld    r0,(gp,_Z - .MN.gptop);;
```

このコードは、1命令でアクセスできる範囲を超えているため、リンカにより複数命令に展開される。よって、1命令アクセスにはならない。

【0091】

なお、gp領域の1命令アクセス範囲は、最大14ビット範囲であるが、オブジェクトの型サイズにより、その範囲は異なる。つまり、8バイト型であれば14ビット範囲であり、4バイト型であれば13ビット範囲であり、2バイト型であれば12ビット範囲であり、1バイト型であれば11ビット範囲である。

【0092】

コンパイラ100は、最大データサイズ（デフォルト32バイト）以下の配列・構造体の実体をgp領域に配置する。一方、グローバル領域配置の最大データサイズを超えるオブジェクトに関しては、gp領域以外に実体を配置し、gp領域にはオブジェクトの先頭アドレスのみを配置する。

【0093】

ここで、gp領域に余裕があれば、データサイズ32バイト以上のオブジェクトも配置させた方が、より良いコードを生成することが可能である。

【0094】

そこで、以下のオプションを用いることで、ユーザは、この最大データサイズを任意の値に指定することが可能となっている。

・コンパイルオプション

```
-mmax-gp-datasize=NUM
```



ここで、NUMは、グローバル領域に配置できる、一つの配列および構造体の最大データサイズの指定バイト（デフォルト32バイト）である。

## 【0095】

図39は、グローバル領域割り付け部121の動作を示すフローチャートである。解析部110によって上記オプションが検出された場合には（ステップS100、S101）、グローバル領域割り付け部121は、ソースプログラム101で宣言されている指定されたサイズ（NUMバイト）以下の全ての変数（配列）については、グローバル領域に配置し、NUMバイトを超える変数については、その先頭アドレスだけをグローバル領域に配置し、その実体をグローバル領域以外のメモリ領域に配置する（ステップS102）。このオプションによって、速度向上とサイズ削減という最適化が可能となる。

## 【0096】

なお、この-mmax-gp-datasizeオプションでは、変数個々の配置指定はできない。個々の変数に対してgp領域に配置する/しないを指定するには、後述する#pragma \_gp\_access指令を使用すればよい。また、外部変数・静的変数は、可能な限り1命令アクセス可能なgp領域に配置することが好ましい。さらに、extern宣言を使用して、他のファイルで定義される外部変数にアクセスする場合は、その外部変数のサイズを省略せずに、明記することが好ましい。例えば、外部変数の定義が、

```
int a[8];
```

とされている場合、使用するファイルでは、

```
extern int a[8];
```

と宣言することが好ましい。

## 【0097】

なお、extern宣言外部変数に#pragma \_gp\_access指令を使用する場合、定義の指定（配置する領域）と使用側の指定（アクセスする方法）を必ず合わせる必要がある。

## 【0098】

次に、このようなオプション用いることによる最適化の具体例を示す。

図40は、最大データサイズを変更した場合の最適化の具体例を示す図である。つまり、デフォルトの状態で作成し、その結果、まだ *g p* 領域に空きがあるので、以下のコマンド例のように、最大データサイズを変更して作成した場合における両ケースで得られる生成コードの例が示されている。

```
c:¥>ammp-cc -O -mmax-gp-datasize=40 sample.c
```

【0099】

ここでは、配列 *c* のオブジェクトサイズが、40バイトであるとする。本図の左欄は、デフォルトの状態で作成された場合に生成されるコードの例であり、本図の右欄は、最大データサイズを40に変更して作成された場合に生成されるコードの例である。なお、本図の最上段、中上段、中下段、最下段は、それぞれ、各欄のタイトル、サンプルプログラム（ソースプログラム101）、そこから生成されるコード（機械語プログラム102）、その生成コードのサイクル数及びコードサイズを示している（以下、最適化の具体例を示す他の図についても同様）。

【0100】

本図の左欄の生成コードから分かるように、配列 *c* の実体が *g p* 領域以外に配置されているため、複数命令での絶対アドレスアクセスになっている。一方、本図の右欄の生成コードから分かるように、配列 *c* の実体が *g p* 領域に置かれるように最大データサイズ（40）が指定されたので、配列 *c* の実体が *g p* 領域に配置され、1命令アクセスでの *g p* 相対アクセスになり、実行速度が向上される。つまり、デフォルトでは、10サイクルで実行される8バイトのコードが生成されるのに対し、最大データサイズの変更によって、7サイクルで実行される5バイトのコードが生成される。

【0101】

もう一つの具体例として、ファイル外定義の外部変数の場合の具体例を図41に示す。ここでは、*g p* 領域配置の最大データサイズは40であるとする。本図の左欄は、ファイル外定義の外部変数に関してサイズ指定がない場合に生成されるコードの例を示し、本図の右欄は、サイズ指定がある場合に生成されるコードの例を示している。

## 【0102】

本図の左欄に示されるように、外部定義配列 a と外部定義配列 c のサイズが共に不明なので、g p 領域に配置されているのか否かコンパイラ 100 では判断できず、複数命令での絶対アドレスアクセスのコードが生成されている。

## 【0103】

一方、本図の右欄に示されるように、配列 a の定義サイズが 40 バイト以下なので、g p 領域に実体を配置し、グローバルポインタレジスタ (gp) を用いて 1 命令での g p 相対アクセスのコードが生成されている。また、外部定義配列 c のサイズも明示的に指定されており、g p 領域配置の最大データサイズ以下であるため、配列 c の実体が g p 領域に配置されているものとし、g p 相対アクセスのコードが生成されている。このように、ファイル外定義の外部変数のサイズを指定しない場合には、10 サイクルで実行される 12 バイトのコードが生成されるのに対し、最大データサイズの変更と外部変数のサイズ指定とを行った場合には、7 サイクルで実行される 5 バイトのコードが生成される。

## 【0104】

次に、グローバル領域割り付け部 121 による、(2) グローバル領域配置の指定に関する最適化について説明する。

前述の、グローバル領域配置の最大データサイズ指定 (-mmax-gp-datasize オプション) では、最大データサイズでのみ、g p 領域の配置を指定するため、期待しない変数まで g p 領域に配置されることがある。

## 【0105】

そこで、変数ごとに g p 領域の配置を指定する #pragma 指令が用意されている。

## ・ #pragma 指令

#pragma \_no\_gp\_access 変数名 [, 変数名, ...]

#pragma \_gp\_access 変数名 [, 変数名, ...]

ここで、[] 内は省略可能を意味する。複数指定する場合、"," (カンマ) で変数名を区切れればよい。なお、オプションとプリAGMA指令とが重複又は矛盾した場合は、プリAGMA指令が優先する。

## 【0106】

このようなプラグマ指令に対して、コンパイラ100は次のように動作する。つまり、図39において、解析部110によってプラグマ指令「`#pragma _no_gp_access` 変数名 [,変数名,...]」が検出された場合には(S100、S101)、グローバル領域割り付け部121は、ここで指定された変数については、オプション指定にかかわらず、グローバル領域に配置させないコードを生成し(ステップS103)、一方、解析部110によってプラグマ指令「`#pragma _gp_access` 変数名 [,変数名,...]」が検出された場合には(S100、S101)、グローバル領域割り付け部121は、ここで指定された変数については、オプション指定にかかわらず、グローバル領域に配置させるコードを生成する(ステップS104)。これらの`#pragma`指令によって、速度向上とサイズ削減という最適化が可能となる。

## 【0107】

なお、`#pragma _no_gp_access`指令が指定された場合は、グローバル領域割り付け部121は、その変数については、`gp`領域に実体もアドレスも配置しない。また、最大データサイズ指定よりも、`#pragma _gp_access`指令の方を優先する。もし、同一の変数に関して異なる指定が現れた場合には、コンパイラ100の動作は不定となる。外部変数・静的変数については、可能な限り1命令アクセス可能な`gp`領域に配置することが好ましい。

## 【0108】

次に、このようなプラグマ指令を用いることによる最適化の具体例を示す。`#pragma _gp_access`指令を使用すると、`gp`領域配置の最大データサイズ以上の外部変数・静的変数を`gp`領域に配置させることができるので、その好適例を示す。

## 【0109】

図42は、`#pragma _no_gp_access`指令を用いた場合に生成されるコードの例(左欄)と、`#pragma _gp_access`指令を用いた場合に生成されるコードの例(右欄)とを示す図である。

## 【0110】

本図の左欄に示されるように、配列`c`のサイズは40バイトなのでデフォルト

の場合、先頭アドレスのみ *gp* 領域に配置され、実体は *gp* 領域に配置されない。また、外部定義されている配列 *a* のサイズが 32 バイトなのでデフォルトの場合は、*gp* 領域に実体が配置されているとコンパイラ 100 は判断する。

【0111】

しかし、`#pragma _no_gp_access` 指令により、配列 *c* については、*gp* 領域には先頭アドレスも実体も配置されずに、*gp* 領域以外に実体が配置され、絶対アドレスアクセスのコードが生成される。外部定義の配列 *a* についても、*gp* 領域以外に実体が配置されているとして、絶対アドレスアクセスのコードが生成される。

【0112】

一方、本図の右欄に示されるように、配列 *c* はサイズが 40 バイトなので、デフォルトの場合では *gp* 領域に配置されないが、`#pragma _gp_access` 指令により配列 *c* の実体が *gp* 領域に配置される。ファイル外定義の配列 *a* は、サイズが不明であるが `#pragma` 指令により、*gp* 領域に配置されているものとし、*gp* 相対アクセスコードが生成される。

【0113】

このように、`#pragma _no_gp_access` 指令を用いた場合には、10 サイクルで実行される 12 バイトのコードが生成されるのに対し、`#pragma _gp_access` 指令を用いた場合には、7 サイクルで実行される 5 バイトのコードが生成される。

【0114】

なお、`extern` 宣言外部変数に `#pragma _gp_access` 指令を使用する場合、定義の指定（配置する領域）と使用側の指定（アクセスする方法）を必ず合わせておくことが好ましい。

【0115】

[ソフトウェアパイプライン部 122]

次に、ソフトウェアパイプライン部 122 の動作とその意義について説明する。

ソフトウェアパイプライン最適化は、ループ高速化手法の 1 つである。この最適化が行われると、ループ構造がプロログ部、カーネル部、エピログ部に

変換される。なお、ソフトウェアパイプライン最適化は、それによって実行速度が向上されると判断された場合に行われる。カーネル部は、各イタレーション（繰り返し）をその前後のイタレーションとオーバーラップさせる。これにより、1イタレーション毎の平均処理時間が削減される。

## 【0116】

図43(a)は、ループ処理におけるプロログ部・カーネル部・エピログ部の概念図である。ここでは、イタレーション間で依存関係のない命令X、Y、Zが5回繰り返される場合の命令コード、実行イメージ、生成コードイメージの例が示されている。なお、ループ処理とは、for文、while文、do文等による繰り返し処理である。

## 【0117】

ここで、プロログ部、エピログ部は、可能であれば、図43(b)及び(c)のプロセスに示されるように、除去される。しかし、不可能であれば除去されず、コードサイズが増加することがある。そのため、ソフトウェアパイプライン最適化の動作を指定するオプション及び#pragma指令が用意されている。

## 【0118】

図43(b)は、ループ処理におけるプロログ部・エピログ部を除去するための処理を示す概念図である。つまり、図43(a)で示されたイタレーション間で依存関係のない命令X、Y、Zの5回の繰り返しについて、前記のプロログ部・カーネル部・エピログ部の概念図で示したループの生成コードイメージを並び替えたものが示されている。ただし、図中の[]のついた命令は、読み込まれるが実行されないとする。

## 【0119】

このようにすると、プロログ部・エピログ部は、カーネル部と同じ命令並びになることがわかる。よって、ループ回数は、プロログ部・エピログ部の実行分(4回)だけ増えるが、[]のついた命令をプレディケート(実行条件)によって制御することにより、図43(c)に示されるように、カーネル部だけでコードを生成することができる。

## 【0120】

図43(c)に示された生成コードの実行順序は、以下のようになる。

1回目においては、プレディケート[C2]、[C3]の付加された命令は実行されない。よって、[C4]Xのみ実行される。

2回目においては、プレディケート[C2]の付加された命令は実行されない。よって、[C3]Y、[C4]Xのみ実行される。

3～5回目においては、[C2]Z、[C3]Y、[C4]Xすべてが実行される。

6回目においては、プレディケート[C4]の付加された命令は実行されない。よって、[C2]Zと[C3]Yのみ実行される。

7回目においては、プレディケート[C3]、[C4]の付加された命令は実行されない。よって、[C2]Zのみ実行される。

【0121】

このように、カーネル部のループ1回目、2回目でプロログ部を、6回目、7回目でエピログ部を実行していることになる。

【0122】

よって、プロログ部・エピログ部のあるループでは、コードサイズが増加するが、ループ回数が減少するため、実行速度向上を期待できる。逆に、プロログ部・エピログ部を除去したループでは、コードサイズを削減できるが、ループ回数が増加するため、実行サイクル数が増加する。

【0123】

そこで、このような最適化の選択を指定可能にするために、以下のコンパイルオプションとプラグマ指令が用意されている。

・コンパイルオプション

`-fno-software-pipelining`

・#pragma指令

`#pragma _no_software_pipelining`

`#pragma _software_pipelining_no_proepi`

`#pragma _software_pipelining_with_proepi`

なお、オプションとプラグマ指令が重複又は矛盾した場合には、プラグマ指令が優先する。

## 【0124】

図44は、ソフトウェアパイプライン部122の動作を示すフローチャートである。解析部110によってオプション「-fno-software-pipelining」が検出された場合には（ステップS110、S111）、ソフトウェアパイプライン部122は、対象となるソースプログラム101中の全てのループ処理に対してソフトウェアパイプライン最適化を行わない（ステップS112）。このオプションによって、コードサイズが増加してしまうことが回避される。

## 【0125】

また、解析部110によってプリAGMA指令「#pragma \_no\_software\_pipelining」が検出された場合には（ステップS110、S111）、ソフトウェアパイプライン部122は、オプション指定にかかわらず、この指定の直後に置かれている1つのループ処理について、ソフトウェアパイプライン最適化を行わない（ステップS113）。これによって、コードサイズが削減される。

## 【0126】

また、解析部110によってプリAGMA指令「#pragma \_software\_pipelining\_no\_proepi」が検出された場合には（ステップS110、S111）、ソフトウェアパイプライン部122は、オプション指定にかかわらず、この指定の直後に置かれている1つのループ処理について、プロログ部・エピログ部が除去できる範囲でソフトウェアパイプライン最適化を行う（ステップS114）。これによって、速度の向上とサイズの削減化が図られる。

## 【0127】

また、解析部110によってプリAGMA指令「#pragma \_software\_pipelining\_with\_proepi」が検出された場合には（ステップS110、S111）、ソフトウェアパイプライン部122は、オプション指定にかかわらず、この指定の直後に置かれている1つのループ処理について、プロログ部・エピログ部を除去せずに、可能な範囲で、ソフトウェアパイプライン最適化を行う（ステップS115）。これによって、速度が向上される。

## 【0128】

なお、ソフトウェアパイプライン部122は、#pragma \_software\_pipelining



ning\_no\_proepi指令に対しては、プロログ部・エピログ部を除去できる範囲でソフトウェアパイプライン最適化を行うが、#pragma \_software\_pipelining\_with\_proepi指令に対しては、プロログ部・エピログ部の除去が可能であっても、除去しない。プロログ部・エピログ部の除去可能なループであっても、図45に示される例のように、プロログ部・エピログ部の除去を抑制することにより、コードサイズは増加するが、実行速度の向上を期待できるからである。また、後述するように、ループ処理の最低繰り返し回数がソフトウェアパイプラインによって重なり合うイタレーション数以上である場合には、ソフトウェアパイプライン部122は、ソフトウェアパイプラインによる最適化を行う。

## 【0129】

図45は、ソフトウェアパイプライン最適化の例を示す図である。なお、この例では、ソフトウェアパイプライン最適化を行うために、コンパイルオプション -O（実行速度とコードサイズ削減の最適化）をつけてコンパイルされている。

## 【0130】

本図の左欄の中下段に示された機械語プログラム102の例から分かるように、デフォルトのソフトウェアパイプライン最適化が行われた場合には、プロログ部・エピログ部のコードも除去され、ループ回数が101回でカーネル部のサイクル数が2サイクルとなり、合計207サイクルで実行され、ループの性能が向上している。

## 【0131】

一方、本図の右欄の中上段に示されたソースプログラム101から分かるように、左欄のソースプログラム101に対して #pragma \_software\_pipelining\_with\_proepi指令が追加指定され、ループのプロログ部・エピログ部の除去を抑制した例となっている。これにより、右欄の中下段に示された機械語プログラム102の例から分かるように、プロログ部・エピログ部のコードが生成されるため、左側と比べコードサイズが増加しているが、ループ回数が99回に減少しており、カーネル部サイクル数が2サイクルであるため、合計204サイクルで実行され、左欄の場合よりも更に実行速度が向上している。なお、プロログ部・エピ

ログ部が周辺コードと並列実行可能な場合には、プロログ部・エピログ部による速度低下の影響は隠蔽できる。

[ループアンローリング部123]

【0132】

次に、ループアンローリング部123の動作とその意義について説明する。ループアンローリング部123は、大きく分けて、(1)ループアンローリングの指定に関する最適化と、(2)ループの繰り返し回数についての保証に関する最適化とを行う。

【0133】

まず、(1)ループアンローリングの指定に関する最適化について説明する。

ループアンローリング最適化とは、ループ高速化手法の1つである。複数のイタレーションを同時に実行することでループ内の実行を高速化する。ループアンローリング最適化を行うことにより、ldp/stp命令の生成や並列度の向上により、実行速度の向上を図ることができる。しかし、コードサイズが増加することと、場合によっては、レジスタ不足によるスピルが発生し、逆に性能が低下してしまう場合がある。

【0134】

なお、ロードペア（ストアペア）命令（ldp/stp命令）とは、二つのロード命令（ストア命令）を1命令で実現した命令である。また、「スピル」とは、空きレジスタを確保する為に、使用されているレジスタを一時的にスタックに退避させることである。この場合には、レジスタの退避・復帰のためにロード・ストア命令が生成される。

【0135】

このようなループアンローリング最適化の動作を指定するオプション及び#pragma指令が用意されている。

・コンパイルオプション

-fno-loop-unroll

・#pragma指令

#pragma \_loop\_unroll

```
#pragma _no_loop_unroll
```

【0136】

図46は、ループアンローリング部123の動作を示すフローチャートである。解析部110によってオプション「-fno-loop-unroll」が検出された場合には（ステップS120、S121）、ループアンローリング部123は、対象となるソースプログラム101中の全てのループ処理に対してループアンローリング最適化を行わない（ステップS122）。このオプションによって、コードサイズが増加してしまうことが回避される。

【0137】

また、解析部110によってプリAGMA指令「#pragma \_loop\_unroll」が検出された場合には（ステップS120、S121）、ループアンローリング部123は、直後に置かれている1つのループ処理に対してループアンローリング最適化を行う（ステップS123）。これによって、速度が向上される。

【0138】

また、解析部110によってプリAGMA指令「#pragma \_no\_loop\_unroll」が検出された場合には（ステップS120、S121）、ループアンローリング部123は、直後に置かれている1つのループ処理に対してループアンローリング最適化を行わない（ステップS124）。これによってコードサイズの増加が回避される。

【0139】

なお、最適化レベル指定に -O/-Ot（実行速度を優先した最適化）が指定されている場合は、ループアンローリング部123は、ループアンローリング最適化が可能であるなら、デフォルトでループアンローリング最適化を行う。最適化レベル指定に -Os（コードサイズ削減を優先した最適化）が指定されている場合は、ループアンローリング部123は、ループアンローリング最適化を行わない。よって、ユーザは、これらの最適化レベル指定コンパイルオプションと組み合わせて、個々のループのループアンローリング最適化の適用を、#pragma \_no\_loop\_unroll指令及び#pragma \_loop\_unroll 指令で制御することが可能である。

【0140】

図47は、`#pragma _loop_unroll`指令による最適化の例を示す図である。本図の左欄は、最適化レベル指定コンパイルオプション `-O`のみをつけてコンパイルした場合の例であり、本図の右欄は、`#pragma _loop_unroll`指令を組み合わせでコンパイルした場合の例である。

## 【0141】

本図の左欄の中下段に示された機械語プログラム102の例から分かるように、プロログ部・エピログ部が除去されたソフトウェアパイプライン最適化が適用されている。そのため、カーネル部の3命令（2サイクル）が101回実行され、全体として合計207サイクルかかっている。

## 【0142】

一方、右欄の中下段に示された機械語プログラム102の例から分かるように、左側と同様にソフトウェアパイプライン最適化が行われ、プロログ部・エピログ部が削除されている。それに、この右欄の機械語プログラム102では、ループアンローリング最適化により、ループ回数が半減しているため、カーネル部の6命令（2サイクル）が52回実行され、全体として合計110サイクルで実行され、速度が向上している。

## 【0143】

次に、ペアメモリアクセス命令（`ldp/stp`）の生成によりループアンローリング最適化をより効果的に使用方法を示す。

## 【0144】

ループアンローリング最適化では、現在のイタレーションと次のイタレーションを同時に実行するため、以下のような連続する領域のデータのロード・ストアが生成される場合がある。

```
ld r1,(r4);;
```

```
ld r2,(r4,4);;
```

## 【0145】

アクセスするデータが、必ず8バイトアラインされて配置されているならば、以下のようなペアメモリアクセス命令（`ldp`命令）を生成することができる。

```
ldp r1:r2,(r4+);;
```

図48は、ペアメモリアクセス命令 (ldp/stp) の生成によりループアンローリング最適化をより効果的に使用する例を示す図である。ここでは、ソフトウェアパイプライン最適化が適用されている。

【0146】

本図の右欄の例では、中下段に示された機械語プログラム102の例から分かるように、ループアンローリング最適化によりループ回数が半減されている。また、`#pragma _align_local_pointer`指令を使用して、ポインタ変数 `pa`、`pb`が8バイトアラインされているアドレスと明示することにより、ロードペア (ストアペア) 命令が生成される。

【0147】

これらの最適化により、左欄の例では、カーネル部の5命令3サイクルが101回実行され、全体として合計308サイクルであるが、右欄の例では、カーネル部の7命令3サイクルが半分の51回実行され、全体として合計158サイクルで実行され、速度が向上している。

【0148】

次に、ループアンローリング部123による、(2) ループの繰り返し回数の保証に関する最適化について説明する。

プログラムの記述上、コンパイラ100ではループ回数を特定することができない場合、ループ高速化の各最適化を効果的に行うことができない。

【0149】

そこで、ユーザは、下記に示す`#pragma`指令にて、ループ回数の情報を提供することにより、より効果的にソフトウェアパイプライン等のループ高速化の最適化を行わせることができる。

・`#pragma`指令

`#pragma _min_iteration=NUM`

`#pragma _iteration_even`

`#pragma _iteration_odd`

【0150】

図46において、解析部110によってプラグマ指令「`#pragma _min_iterati`

on=NUM」が検出された場合には（ステップS120、S121）、ループアンローリング部123は、直後に置かれている1つのループ処理が最低NUM回繰り返されることを前提に、ループアンローリング最適化を行う（ステップS125）。例えば、例えば、保証された繰り返し最低回数がループアンローリングによる展開数以上である場合に、ループアンローリング部123は、そのループ処理のループアンロールを行う。これによって、速度の向上とサイズの削減が図られる。

#### 【0151】

また、解析部110によってプリAGMA指令「#pragma \_iteration\_even」が検出された場合には（ステップS120、S121）、ループアンローリング部123は、直後に置かれている1つのループ処理が偶数回繰り返されることを前提に、ループアンローリング最適化を行う（ステップS126）。これによって、実行速度が向上される。

#### 【0152】

また、解析部110によってプリAGMA指令「#pragma \_iteration\_odd」が検出された場合には（ステップS120、S121）、ループアンローリング部123は、直後に置かれている1つのループ処理が奇数回繰り返されることを前提に、ループアンローリング最適化を行う（ステップS126）。これによって、実行速度が向上される。

#### 【0153】

なお、#pragma \_min\_iteration指令で1以上の値を指定した場合、1回もループを通らない場合のために生成されるエスケープコードを除去できるという効果もある。また、繰り返し回数が不明なループに対して、ループアンローリング最適化を期待する場合、偶数回ループか奇数回ループかが決まっているならば、\_iteration\_even / #pragma \_iteration\_odd指令を使用することにより、ループアンローリング最適化の適用が可能になるため、実行速度向上を期待することができる。

#### 【0154】

図49は、#pragma \_min\_iteration指令による最適化の例を示す図である。こ

ここでは、繰り返し回数が不明なループでの、`#pragma _min_iteration`指令の使用効果が示されている。ただし、サイクル比較のため、引数`end`の値を100とする。

## 【0155】

本図の左欄では、中下段に示された機械語プログラム102の例から分かるように、ループ回数が不明なため、一度もループを実行しない場合にループ本体を飛び越すための `cmple/br` 命令（エスケープコード）が生成されている。また、ループ命令の生成を行うことができないため、加算命令・比較命令・ジャンプ命令でループが生成されている。サイクル数は、ループ部が7命令4サイクルの100回繰り返しとなり、全体として合計405サイクルとなっている。

## 【0156】

一方、本図の右欄では、中上段に示されたソースプログラム101の例から分かるように、繰り返し回数が不明であるが、最低4回繰り返されることが`#pragma _min_iteration`指令で指定されている。これにより、ループ回数が0回の場合を考慮する必要が無いため、ループアンローリング部123は、エスケープコードを生成する必要がなくなる。

## 【0157】

また、ループ最低回数を考慮して、ループアンローリング部123は、ループ命令を生成することができる。例えば、保証された繰り返し最低回数（4）がループアンローリングによる展開数（この例では、3サイクル）以上であるので、ループアンローリング部123は、ループアンロールを行う。

## 【0158】

さらに、この例では、さらにソフトウェアパイプライン最適化が可能になっている。これは、保証されたループの繰り返し最低回数（4）がソフトウェアパイプラインによって重なり合うイタレーション数以上であったために、ソフトウェアパイプライン部122がソフトウェアパイプラインによる最適化を行ったためである。

## 【0159】

右欄の中下段に示された機械語プログラム102の例から分かるように、サイ

クル数は、ループ部が5命令3サイクルの101回繰り返しとなり、全体として合計308サイクルとなり、実行速度とサイズ削減が実現されている。

## 【0160】

図50及び図51は、`#pragma iteration_even`/`#pragma _iteration_odd` 指令による最適化の例を示す図である。図50は、ループ回数が不明の場合におけるソースプログラム101の例（左欄）と、そこから生成される機械語プログラム102の例（右欄）を示す図である。本図から分かるように、実際のループ回数が不明な場合、ループアンローリング最適化は適用できない。これは、ループ回数が偶数回の場合と奇数回の場合では、ループアンローリング最適化によって生成されるコードが異なるためである。

## 【0161】

ところが、図51に示されるように、繰り返し回数が不明なループの場合でも、偶数回ループか奇数回ループかを指定することにより、ループアンローリング最適化を適用することができる。

## 【0162】

本図の左欄では、ループ回数が偶数回であることを、`#pragma _iteration_even`指令で指定されているため、ループアンローリング部123によるループアンローリング最適化が行われ、左欄の中下段に示された機械語プログラム102の例から分かるように、偶数回用のコードが生成されている。

## 【0163】

また、本図の右欄では、ループ回数が奇数回であることを、`#pragma _iteration_odd`指令により指定されているため、ループアンローリング部123によるループアンローリング最適化が行われ、右欄の中下段に示された機械語プログラム102の例から分かるように、奇数回用のコードが生成されている。この右欄の例から分かるように、左欄に示された偶数回の場合の生成コードと初期化部・ループ部はほぼ同じで、後処理部に、ループの最後の一回分を実行するコードが生成されている。

## 【0164】

このように、ループ回数が不明であっても、偶数回であるか奇数回であるかを



保証することで、ループアンローリング部123は、ループアンローリング最適化を行うことができ、これによって実行速度が向上される。

【0165】

[if変換部124]

次に、if変換部124の動作とその意義について説明する。

通常、C言語プログラムのif構造をコンパイルすると、分岐命令（br命令）が生成される。これに対して、if変換とは、C言語プログラムのif構造を分岐命令を用いることなく、条件付き実行命令だけに書き換えることである。これによって、実行順序が固定化される（順次実行となる）るので、パイプラインの乱れが回避され、実行速度が向上され得る。なお、条件付き実行命令とは、その命令に含まれる条件（プレディケート）がプロセッサ1の状態（コンディションフラグ）と一致している場合にだけ実行される命令である。

【0166】

if変換により、if構造のワーストケースにおける実行時間は短縮されるが、ベストケースにおける実行時間は（短縮後の）ワースト実行時間と等しくなる。そのために、if構造の特性（条件成立・不成立それぞれの発生頻度や各パスの実行サイクル数）に応じて、if変換を適用すべき場合とすべきでない場合がある。

【0167】

このため、ユーザは、適用の可否をコンパイルオプションや#pragma指令で指示することができる。

・コンパイルオプション

-fno-if-conversion

・#pragma指令

#pragma \_if\_conversion

#pragma \_no\_if\_conversion

なお、オプションとプリAGMA指令が重複又は矛盾した場合には、プリAGMA指令が優先する。

【0168】

図52は、if変換部124の動作を示すフローチャートである。解析部110によってオプション「-fno-if-conversion」が検出された場合には（ステップS130、S131）、if変換部124は、対象となるソースプログラム101中の全てのif構造文に対してif変換を行わない（ステップS132）。なお、本オプションが検出されない場合は、if変換部124は、if変換が可能であり、かつ、そのワーストケースの時間がif変換前に対して短いif構造文である場合に、そのif構造文をif変換する。

【0169】

また、解析部110によってプリAGMA指令「#pragma\_if\_conversion」が検出された場合には（ステップS130、S131）、if変換部124は、オプション指定にかかわらず、直後に置かれている1つのif構造文に対して、可能であればif変換を行う（ステップS133）。これによって、速度が向上される。

【0170】

また、解析部110によってプリAGMA指令「#pragma\_no\_if\_conversion」が検出された場合には（ステップS130、S131）、if変換部124は、オプション指定にかかわらず、直後に置かれている1つのif構造文に対して、if変換を行わない（ステップS134）。これによって、速度が向上される。

【0171】

図53は、#pragma\_no\_if\_conversion指令でコンパイルした場合と、#pragma\_if\_conversion指令でコンパイルした場合の機械語プログラム102の例を示す図である。

【0172】

本図の左欄では、中下段の機械語プログラム102の例から分かるように、if変換を抑制したことにより、分岐命令が生成されている（実行サイクル数：5あるいは7、コードサイズ：12バイト）。

【0173】

一方、本図の右欄では、中下段の機械語プログラム102の例から分かるように、#pragma指令によってif変換を行うこととしたことにより、分岐命令が、

条件付き命令（プレディケート付き命令）に置き換わっている（実行サイクル数：4、コードサイズ：8バイト）。このように、if変換を実施することで、実行速度比1.25倍、コードサイズ比67%が達成されている。

【0174】

[ペア命令生成部125]

次に、ペア命令生成部125の動作とその意義について説明する。ペア命令生成部125は、大きく分けて、（1）配列・構造体のアラインメントの設定に関する最適化と、（2）仮引数ポインタ・ローカルポインタのアラインの保証に関する最適化とを行う。

【0175】

まず、（1）配列・構造体のアラインメントの設定に関する最適化について説明する。

ユーザは、以下のオプションを用いて、配列と構造体の先頭アドレスのアラインを指定することができる。アラインメントを調整することで、メモリアクセス命令のペアリング（2つのレジスタとメモリ間の転送をひとつの命令で行うこと）が可能となり、実行速度の向上が期待できる。その反面、アラインメント値を大きくすると、データの未使用領域が増加し、データサイズが増大する可能性がある。

・コンパイルオプション

-falign\_char\_array=NUM (NUM=2,4または8)

-falign\_short\_array=NUM (NUM=4または8)

-falign\_int\_array=NUM (NUM=8)

-falign\_all\_array=NUM (NUM=2,4または8)

-falign\_struct=NUM (NUM=2,4または8)

【0176】

上記オプションは、上から順に、char型の配列、short型整数、int型整数、それら3つのデータ型全ての配列、構造体のアラインメントを指定している。また、“NUM”は、アラインするサイズ（バイト）を示す。

【0177】

図54は、ペア命令生成部125の動作を示すフローチャートである。解析部110によって上記オプションのいずれかが検出された場合には（ステップS140、S141）、ペア命令生成部125は、対象となるソースプログラム101で宣言されている指定された型の全ての配列又は構造体について、その先頭アドレスが指定されたNUMバイトのアラインとなるように配列又は構造体をメモリに配置し、その配列又は構造体にアクセスする命令については、可能な場合に、ペアリング（2つのレジスタとメモリ間の転送を並行して行う命令の生成）を行う（ステップS142）。これによって、実行速度が向上される。

【0178】

図55は、サンプルプログラムをオプションなしでコンパイルした場合と、オプション'-falign-short-array=4'でコンパイルした場合のアセンブリコードを示す図である。

【0179】

本図の左欄に示されたオプションなしの場合、中下段に示された機械語プログラム102の例から分かるように、アラインメントが不明のため、ロード命令のペアリング（2つのレジスタとメモリ間の転送をひとつの命令で行う）ができない（実行サイクル数：25、コードサイズ：22）。

【0180】

一方、本図の右欄に示されたオプションありの場合、配列が4バイトでアラインされるため、中下段に示された機械語プログラム102の例から分かるように、最適化部120によるペアリングが実現されている（実行サイクル数：15、コードサイズ：18）。このように、アラインメントの指定によって、実行速度比1.67倍、コードサイズ比82%が達成されている。

【0181】

次に、ペア命令生成部125による、（2）仮引数ポインタ・ローカルポインタのアラインの保証に関する最適化を説明する。

【0182】

ユーザは、以下のプリAGMA指令を用いて、関数引数のポインタ変数の指すデータのアラインメントや、ローカルポインタ変数の指すデータのアラインメントを

保証することで、最適化部120によるメモリアクセス命令のペアリングが可能となり、実行速度の向上が期待できる。

・#pragma指令

```
#pragma _align_parm_pointer=NUM 変数名 [, 変数名, ...]
```

```
#pragma _align_local_pointer=NUM 変数名 [, 変数名, ...]
```

【0183】

なお、“NUM”はアラインするサイズ（2，4又は8バイト）を表す。また、上記#pragma指令で保証されたポインタ変数の指すデータが指定されたバイト境界にアラインされていなかった場合には、プログラムの正常動作は保証されない。

【0184】

図54において、解析部110によってプラグマ指令「#pragma \_align\_parm\_pointer=NUM 変数名 [, 変数名, ...]」が検出された場合には（ステップS140、S141）、ペア命令生成部125は、“変数名”で示される引数のポインタ変数の指すデータが引数渡しの時点でNUMバイトにアラインされているものとし、その配列にアクセスする命令については、可能な場合に、ペアリングを行う（ステップS143）。これによって、実行速度が向上される。

【0185】

また、解析部110によってプラグマ指令「#pragma \_align\_local\_pointer=NUM 変数名 [, 変数名, ...]」が検出された場合には（ステップS140、S141）、ペア命令生成部125は、“変数名”で示されるローカルポインタ変数の指すデータが関数内部で常にNUMバイトでアラインされているものとし、その配列にアクセスする命令については、可能な場合に、ペアリングを行う（ステップS144）。これによって、実行速度が向上される。

【0186】

図56は、プラグマ指令「#pragma \_align\_parm\_pointer=NUM 変数名 [, 変数名, ...]」による最適化の例を示す図である。

【0187】

本図の左欄に示されるように、#pragma \_align\_parm\_pointerを与えない場合、ポインタ変数srcの指すデータのアラインメントが不明なため、中下段に示さ

れた機械語プログラム102の例から分かるように、各データはそれぞれ独立にロードされる（実行サイクル数160、コードサイズ：24バイト）。

【0188】

一方、本図の右欄に示されるように、`#pragma`指令を与えると、データは4バイト境界にアラインされるため、中下段の機械語プログラム102の例から分かるように、メモリ読み出しのペアリングが行われる（実行サイクル数：107、コードサイズ：18バイト）。このように、アライメントを指定することで、実行速度比1.50倍、コードサイズ比43%が達成される。

【0189】

図57は、プラグマ指令「`#pragma _align_local_pointer=NUM 変数名 [, 変数名, ...]`」による最適化の例を示す図である。

本図の左欄に示されるように、`#pragma _align_local_pointer`を与えない場合、ポインタ変数`from`、`to`の指すデータのアラインメントが不明なため、中下段に示された機械語プログラム102の例から分かるように、配列要素はそれぞれ独立にロードされる（実行サイクル数：72、コードサイズ：30）。

【0190】

一方、本図の右欄に示されるように、`#pragma _align_parm_pointer`を与えることで、中下段に示された機械語プログラム102の例から分かるように、ポインタ変数`from`、`to`の指すデータが4バイト境界にアラインされていることを利用したメモリ読み出しのペアリングが可能となる。（実行サイクル数：56、コードサイズ：22）。このように、アライメントを指定することで、実行速度比1.32倍、コードサイズ比73%が達成される。

【0191】

【発明の効果】

以上の説明から明らかなように、本発明に係るコンパイラは、変数のグローバル領域への割り付けに関する指示を受け付け、その指示に基づいて、各種変数のグローバル領域へのマッピングを行う。

【0192】

その1つとして、ユーザは、コンパイル時のオプションによって、グローバル

領域に割り付ける変数の最大データサイズを指定することができる。これによって、ユーザは、グローバル領域に配置させる変数のデータサイズを制御することが可能となり、グローバル領域を有効活用するように最適化を図ることができる。

## 【0193】

また、ユーザは、ソースプログラム中に置くプリAGMA指令によって、変数ごとに、グローバル領域への割り付けをする／しない旨の指定をすることができる。これによって、グローバル領域に優先的に割り付けるべき変数、割り付けてはいけない変数を個々に区別して、最適なグローバル領域の割り付けをユーザが管理することができる。

## 【0194】

また、本発明に係るコンパイラは、ソフトウェアパイプラインの指示を受け付け、その指示に従ったソフトウェアパイプラインによる最適化を行う。

## 【0195】

その1つとして、ユーザは、コンパイル時のオプションによって、ソフトウェアパイプラインをしない旨の指定をすることができる。これによって、ソフトウェアパイプラインによるコードサイズの増加が抑制される。またソフトウェアパイプラインが行われたアセンブラコードは複雑なので、プログラムの機能検証のために、ソフトウェアパイプラインを抑制することで、デバッグが容易となる。

## 【0196】

また、ユーザは、ソースプログラム中のプリAGMA指令によって、ループ処理ごとに、ソフトウェアパイプラインをする／しない旨の指定をしたり、プロログ部及びエピログ部を除去して／除去しないでソフトウェアパイプラインをする旨の指定をしたりすることができる。これによって、ループ処理ごとにソフトウェアパイプラインをするか否かを選択したり、コードサイズ重視(プロログエピログ除去する)又はスピード重視(プロログエピログ除去しない)のソフトウェアパイプラインを選択したりすることが可能となる。

## 【0197】

また、本発明に係るコンパイラは、ループアンローリングの指示を受け付け、その指示に従ったループアンローリングによる最適化を行う。

【0198】

その1つとして、ユーザは、コンパイル時のオプションによって、ループアンローリングをしない旨の指示をすることができる。これによって、ループアンローリングによるコードサイズの増加が回避される。

【0199】

また、ユーザは、ソースプログラム中のプリAGMA指令によって、ループ処理ごとに、ループアンローリングをする／しない旨の指示をすることができる。これによって、ユーザは、ループ処理ごとに、その繰り返し回数等を勘案し、実行速度を重視するか、コードサイズを重視するかの最適化を選択することができる。

【0200】

また、本発明に係るコンパイラは、ループ処理の繰り返し回数に関する指示を受け付け、その指示に従った最適化を行う。

【0201】

その1つとして、ユーザは、ソースプログラム中のプリAGMA指令によって、ループ処理ごとに、最低の繰り返し回数を保証することができる。これによって、繰り返し回数が0である場合に必要とされるコード（エスケープコード）を生成する必要がなくなるとともに、ソフトウェアパイプラインニングやループアンロールによる最適化が可能となる。

【0202】

また、ユーザは、ソースプログラム中のプリAGMA指令によって、ループ処理ごとに、繰り返し回数が偶数回／奇数回であることを保証することができる。これによって、たとえ繰り返し回数が不明であっても、ループ処理ごとに、ループアンロールによる最適化が可能となり、実行速度が向上され得る。

【0203】

また、本発明に係るコンパイラは、if変換に関する指示を受け付け、その指示に従ったif変換による最適化を行う。

【0204】



その1つとして、ユーザは、コンパイル時のオプションによって、if変換をしない旨の指示をすることができる。これによって、if構造のthen側とelse側で命令数のバランスが悪い場合に、if変換によって命令数の少ない側の実行が命令数の多い側に制約されてしまうという不具合の発生を防ぐことができる。

【0205】

また、ユーザは、ソースプログラム中のプリAGMA指令によって、ループ処理ごとに、if変換をする／しない旨の指示をすることができる。これによって、個々のループ処理の特性（then側とelse側それぞれの命令数のバランス、予想される発生頻度のバランス等）を考慮して、より実行速度が向上されると予測される選択（if変換をする／しない）を行うことができる。

【0206】

また、本発明に係るコンパイラは、配列データのメモリ領域への配置におけるアライメントに関する指示を受け付け、その指示に従った最適化を行う。

【0207】

その1つとして、ユーザは、コンパイル時のオプションによって、特定の型の配列データについて、バイト数によるアラインを指定することができる。これによって、2つのデータに対するメモリ・レジスタ間の転送を同時に行うペア命令が生成され、実行速度が向上される。

【0208】

また、ユーザは、ソースプログラム中のプリAGMA指令によって、ポインタ変数が指すデータのアラインを指定することができる。これによって、データごとに、ペア命令が生成されることを可能にすることができ、実行速度が向上される。

【0209】

以上のように、本発明に係るコンパイラにより、ユーザは、従来のような大雑把な指示ではなく、コンパイラによる各種最適化の種類ごとにON/OFFやその程度を指定したり、プログラム中の変数やループ処理等の単位で最適化をON/OFFさせたりする等のきめ細かい制御が可能となり、特に、精密な最適化のチューニングが必要とされるメディア処理のアプリケーション開発に有効であり、その実用的価値は極めて高い。

【図面の簡単な説明】

【図 1】 本発明に係るコンパイラの対象となるプロセッサの概略ブロック図である。

【図 2】 同プロセッサの算術論理・比較演算器の概略図を示す。

【図 3】 同プロセッサのバレルシタの構成を示すブロック図である。

【図 4】 同プロセッサの変換器の構成を示すブロック図である。

【図 5】 同プロセッサの除算器の構成を示すブロック図である。

【図 6】 同プロセッサの乗算・積和演算器の構成を示すブロック図である。

【図 7】 同プロセッサの命令制御部の構成を示すブロック図である。

【図 8】 同プロセッサの汎用レジスタ（R0～R31）の構造を示す図である。

【図 9】 同プロセッサのリンクレジスタ（LR）の構造を示す図である。

【図 10】 同プロセッサの分岐レジスタ（TAR）の構造を示す図である。

【図 11】 同プロセッサのプログラム状態レジスタ（PSR）の構造を示す図である。

【図 12】 同プロセッサの条件フラグレジスタ（CFR）の構造を示す図である。

【図 13】 同プロセッサのアキュムレータ（M0, M1）の構造を示す図である。

【図 14】 同プロセッサのプログラムカウンタ（PC）の構造を示す図である。

【図 15】 同プロセッサのPC退避用レジスタ（IPC）の構造を示す図である。

【図 16】 同プロセッサのPSR退避用レジスタ（IPSR）の構造を示す図である。

【図 17】 同プロセッサのパイプライン動作を示すタイミング図である。

【図 18】 同プロセッサによる命令実行時の各パイプライン動作を示すタ

イメージ図である。

【図 19】 同プロセッサの並列動作を示す図である。

【図 20】 同プロセッサが実行する命令のフォーマットを示す図である。

【図 21】 カテゴリー「ALUadd（加算）系」に属する命令を説明する図である。

【図 22】 カテゴリー「ALUsub（減算）系」に属する命令を説明する図である。

【図 23】 カテゴリー「ALUlogic（論理演算）系ほか」に属する命令を説明する図である。

【図 24】 カテゴリー「CMP（比較演算）系」に属する命令を説明する図である。

【図 25】 カテゴリー「mul（乗算）系」に属する命令を説明する図である。

【図 26】 カテゴリー「mac（積和演算）系」に属する命令を説明する図である。

【図 27】 カテゴリー「msu（積差演算）系」に属する命令を説明する図である。

【図 28】 カテゴリー「MEMld（メモリ読み出し）系」に属する命令を説明する図である。

【図 29】 カテゴリー「MEMstore（メモリ書き出し）系」に属する命令を説明する図である。

【図 30】 カテゴリー「BRA（分岐）系」に属する命令を説明する図である。

【図 31】 カテゴリー「BSasl（算術バレルシフト）系ほか」に属する命令を説明する図である。

【図 32】 カテゴリー「BSlsr（論理バレルシフト）系ほか」に属する命令を説明する図である。

【図 33】 カテゴリー「CNVvaln（算術変換）系」に属する命令を説明する図である。

【図 3 4】 カテゴリー「CNV（一般変換）系」に属する命令を説明する図である。

【図 3 5】 カテゴリー「SATvlpk（飽和处理）系」に属する命令を説明する図である。

【図 3 6】 カテゴリー「ETC（その他）系」に属する命令を説明する図である。

【図 3 7】

本発明に係るコンパイラの構成を示す機能ブロック図である。

【図 3 8】

（a）は、グローバル領域におけるデータ等の配置例、（b）は、グローバル領域以外の領域におけるデータの配置例を示す図である。

【図 3 9】

グローバル領域割り付け部の動作を示すフローチャートである。

【図 4 0】

グローバル領域割り付け部による最適化の具体例を示す図である。

【図 4 1】

グローバル領域割り付け部による最適化の具体例を示す図である。

【図 4 2】

グローバル領域割り付け部による最適化の具体例を示す図である。

【図 4 3】

ソフトウェアパイプライン最適化を説明する図である。

【図 4 4】

ソフトウェアパイプライン部の動作を示すフローチャートである。

【図 4 5】

ソフトウェアパイプライン部による最適化の具体例を示す図である。

【図 4 6】

ループアンローリング部の動作を示すフローチャートである。

【図 4 7】

ループアンローリング部による最適化の具体例を示す図である。

【図 4 8】

ループアンローリング部による最適化の具体例を示す図である。

【図 4 9】

ループアンローリング部による最適化の具体例を示す図である。

【図 5 0】

ループアンローリング最適化が適用できない例を示す図である。

【図 5 1】

ループアンローリング部による最適化の具体例を示す図である。

【図 5 2】

i f 変換部の動作を示すフローチャートである。

【図 5 3】

i f 変換部による最適化の具体例を示す図である。

【図 5 4】

ペア命令生成部の動作を示すフローチャートである。

【図 5 5】

ペア命令生成部による最適化の具体例を示す図である。

【図 5 6】

ペア命令生成部による最適化の具体例を示す図である。

【図 5 7】

ペア命令生成部による最適化の具体例を示す図である。

【符号の説明】

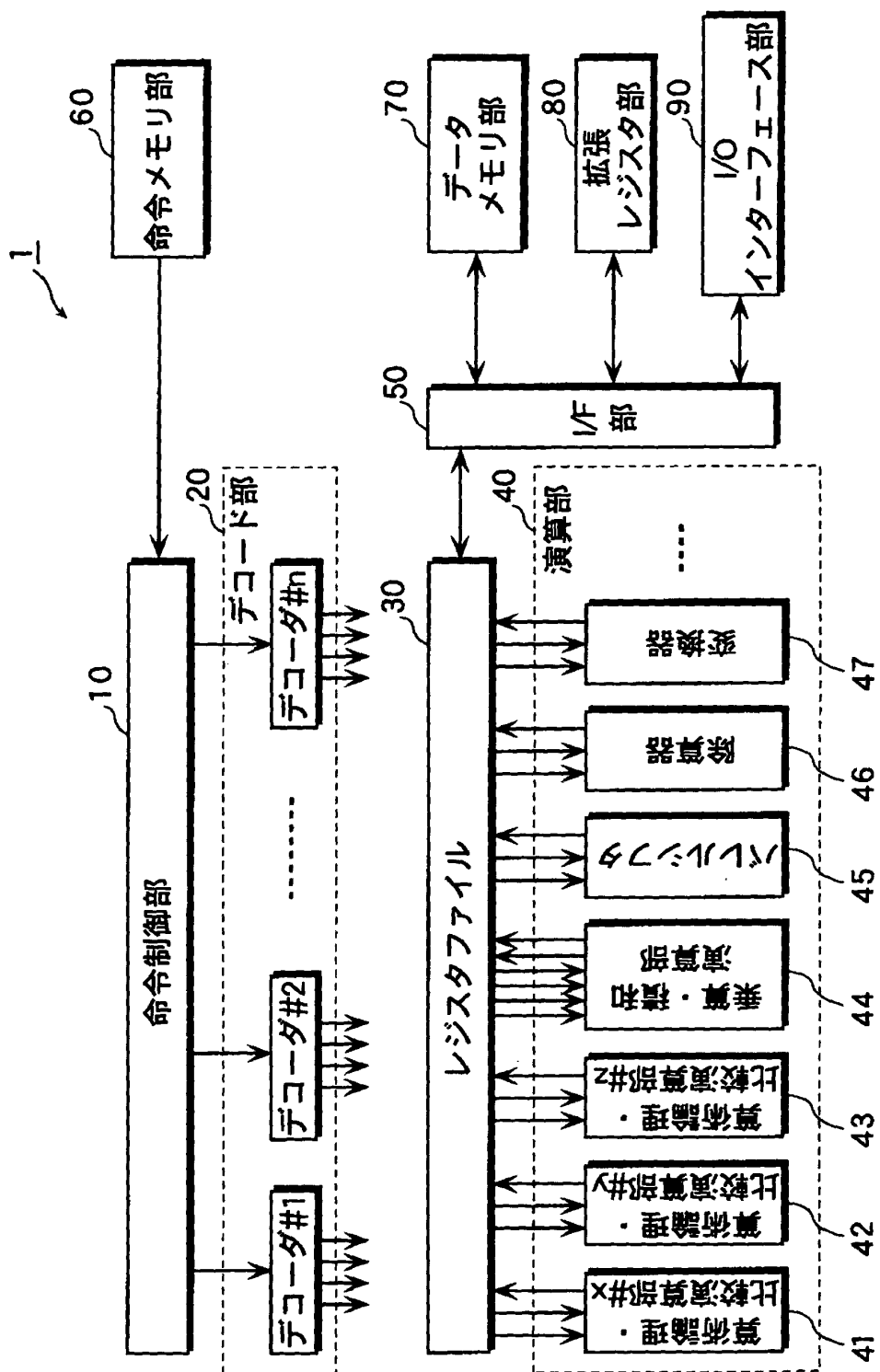
- 1     プロセッサ
- 1 0   命令制御部
- 2 0   デコード部
- 3 0   レジスタファイル
- 3 1   プログラム状態レジスタ (P S R)
- 3 2   条件フラグレジスタ (C F R)
- 3 3   プログラムカウンタ (P C)
- 3 4   P C 退避用レジスタ (I P C)

- 3 5     P S R 退避用レジスタ ( I P S R )
- 4 0     演算部
- 4 1 ~ 4 3   算術論理・比較演算器
- 4 4     積和演算器
- 4 5     バレルシフタ
- 4 6     除算器
- 4 7     変換器
- 5 0     I / F 部
- 6 0     命令メモリ部
- 7 0     データメモリ部
- 8 0     拡張レジスタ部
- 9 0     I / O インターフェース部
- 1 0 0    コンパイラ
- 1 0 1    ソースプログラム
- 1 0 2    機械語プログラム
- 1 1 0    解析部
- 1 1 2    ステップ S
- 1 2 0    最適化部
- 1 2 1    グローバル領域割り付け部
- 1 2 2    ソフトウェアパイプラインニング部
- 1 2 3    ループアンローリング部
- 1 2 4    i f 変換部
- 1 2 5    ペア命令生成部
- 1 3 0    出力部

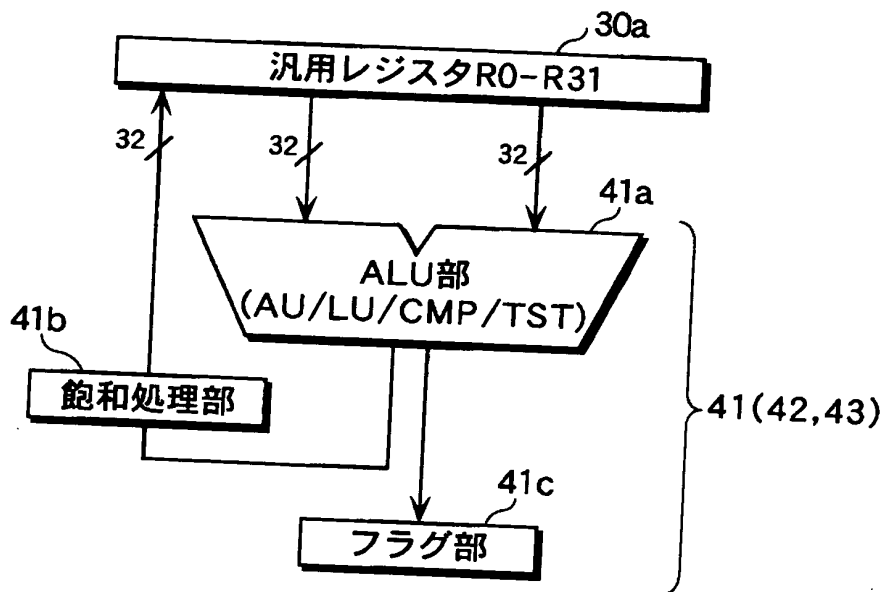
【書類名】

図面

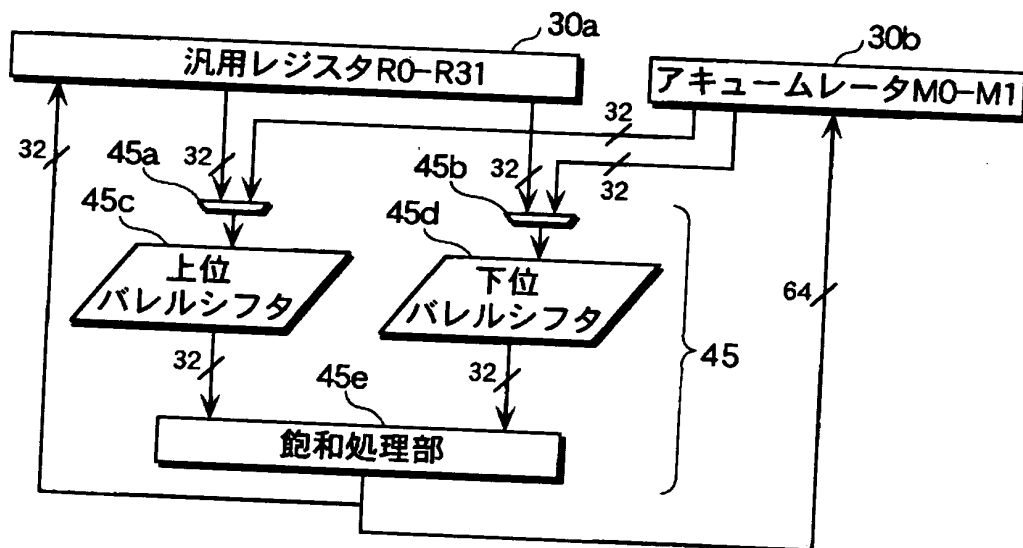
【図 1】



【図 2】

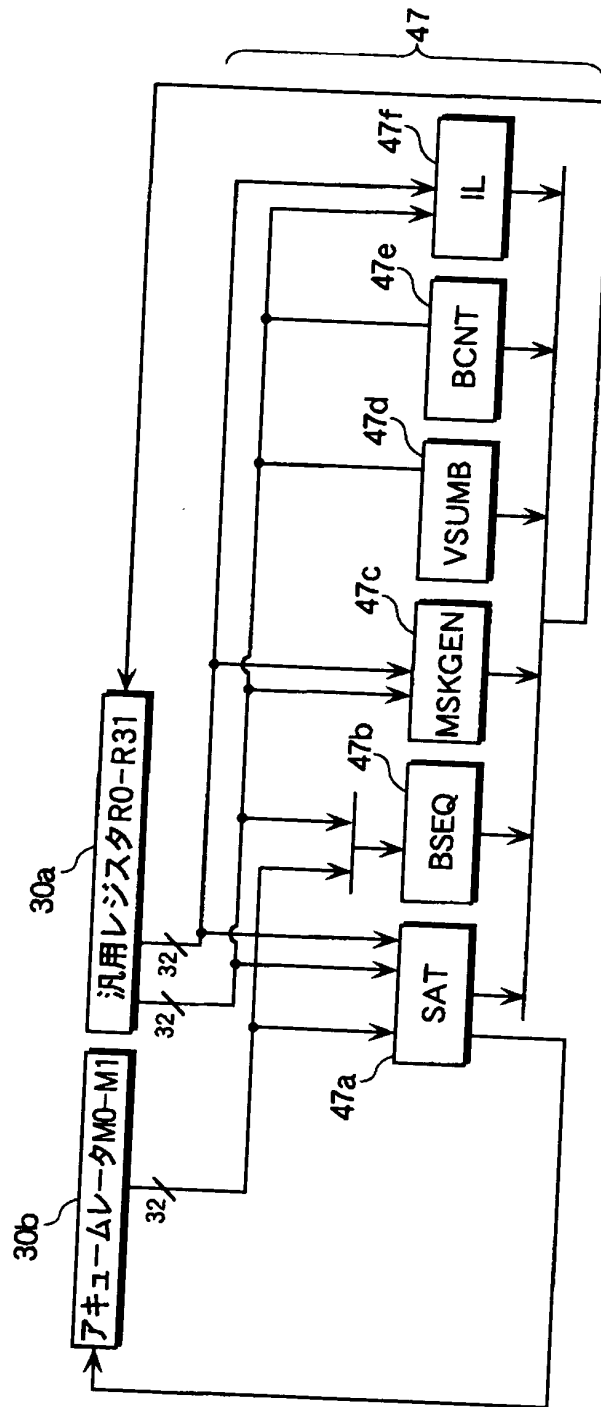


【図 3】

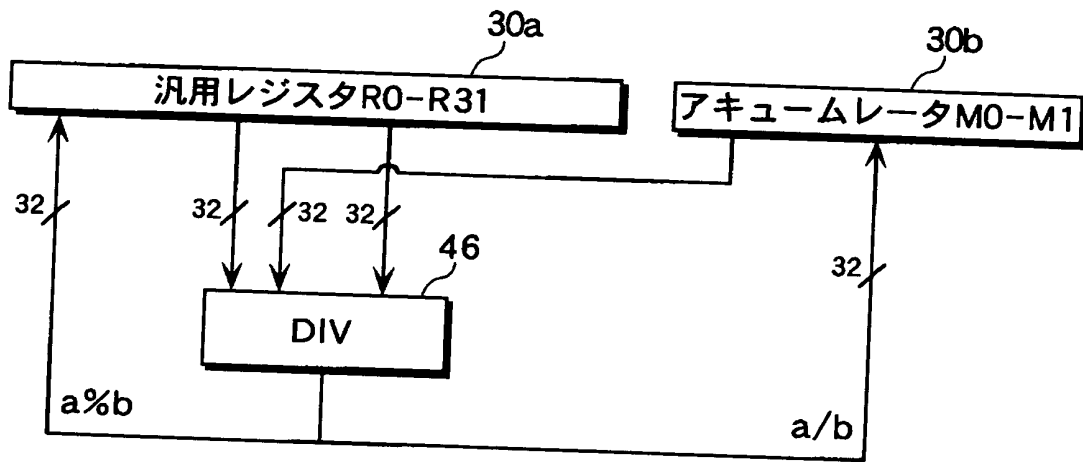




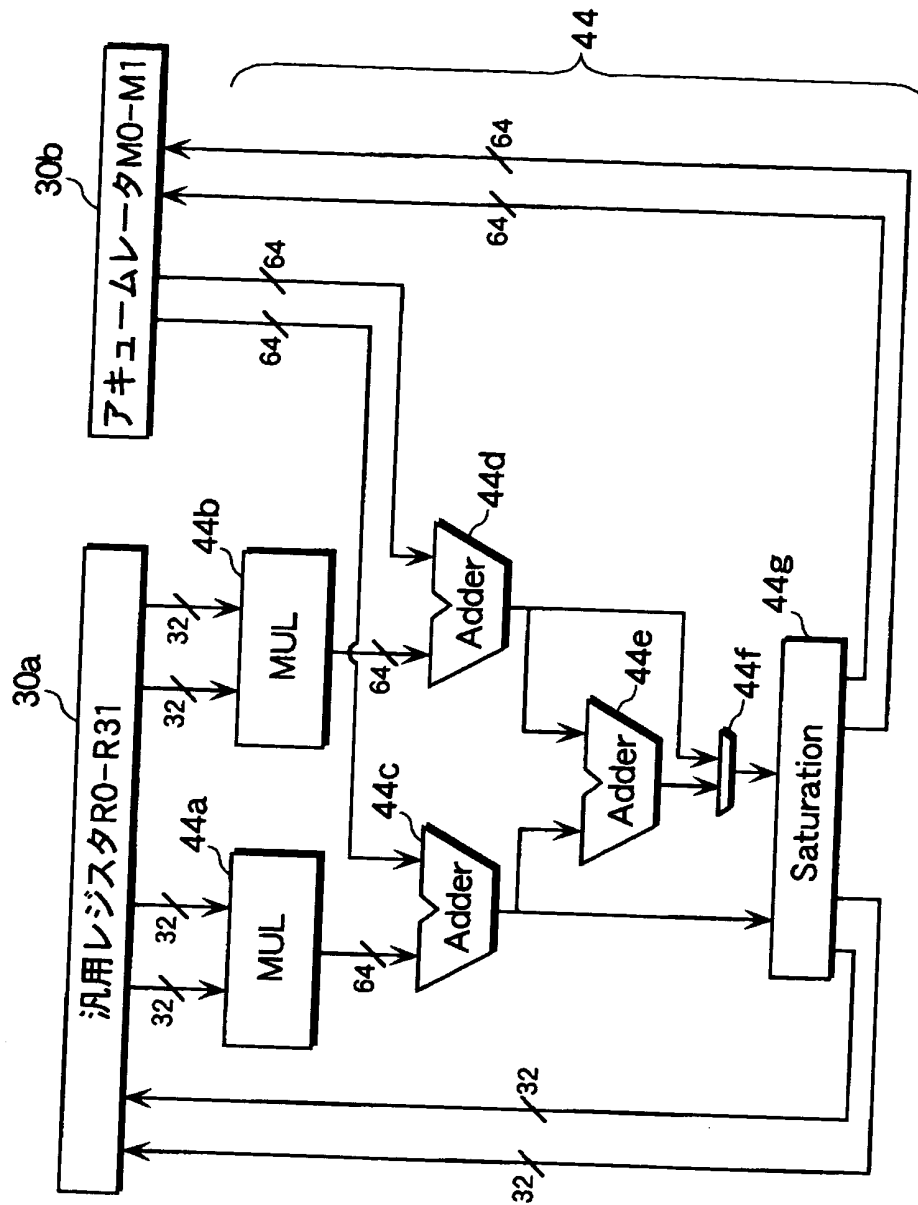
【図4】



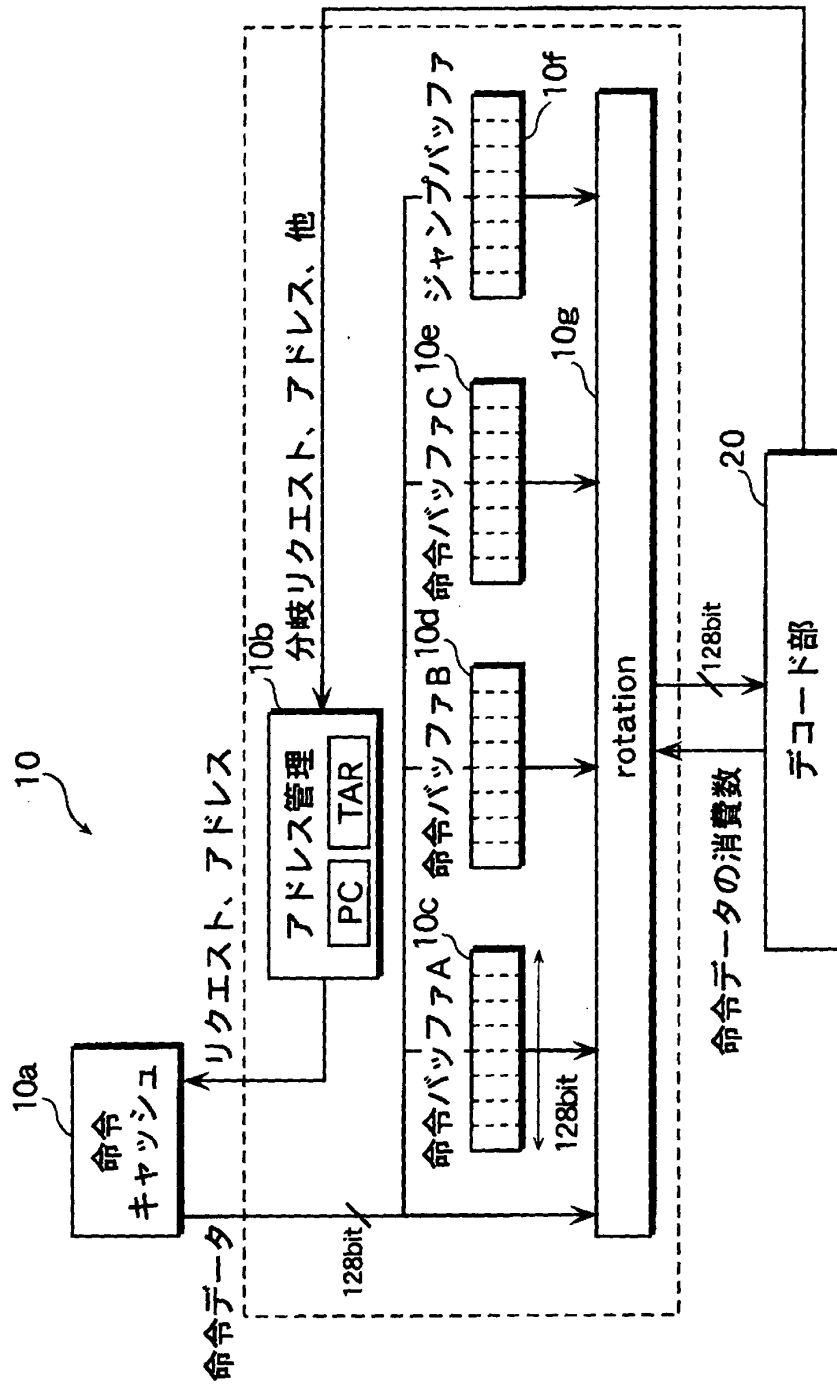
【図5】



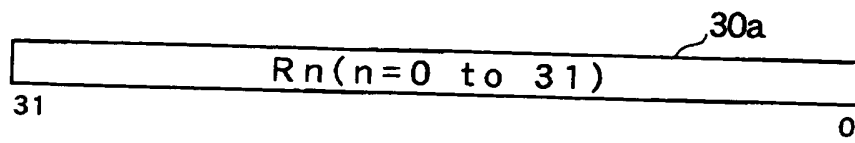
【図 6】



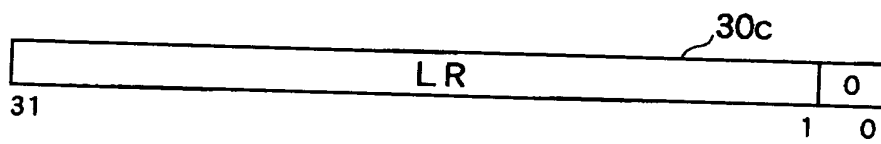
【図7】



【図 8】



【図 9】



【図 1 0】



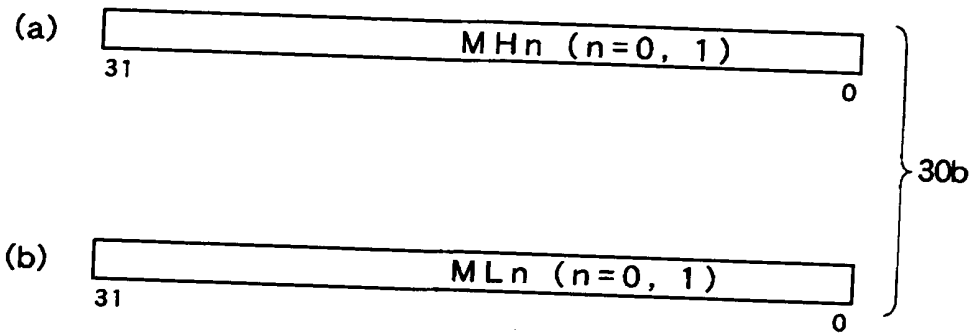
【図11】

ビット	31	30	29	28	27	26	25	24	PL	
ビット名	reserved	SWE	FXP	reserved	IH	EH				
ビット	23	22	21	20	19	18	17	16	IE	
ビット名	LPIE3	LPIE2	LPIE1	LPIE0	reserved	reserved	AEE			
ビット	15	14	13	12	11	10	9	8	Reserved	
ビット名										
ビット	7	6	5	4	3	2	1	0	IM[7:0]	
ビット名										

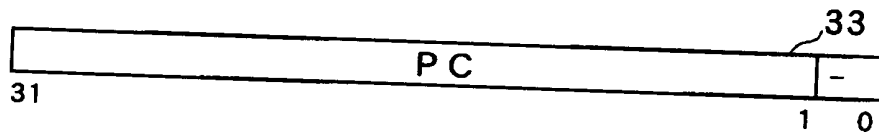
【図12】

ビット	31	30	29	28	27	26	25	24
ビット名	ALN		reserved	BPO				
ビット	23	22	21	20	19	18	17	16
ビット名	reserved			VC3		VC2	VC1	VC0
ビット	15	14	13	12	11	10	9	8
ビット名	reserved							OVS
ビット	7	6	5	4	3	2	1	0
ビット名	C7	C6	C5	C4	C3	C2	C1	C0

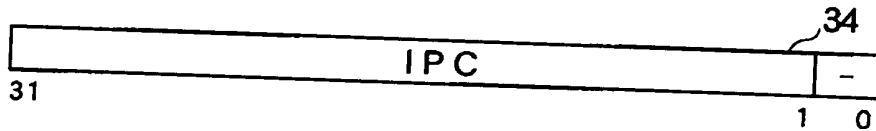
【図 13】



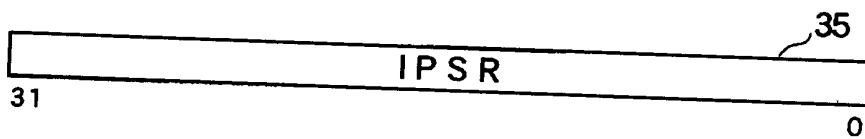
【図 14】



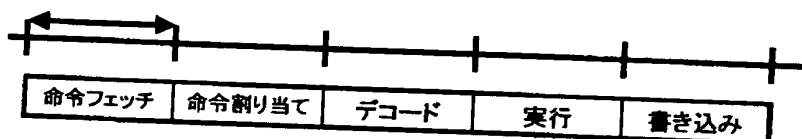
【図 15】



【図 16】

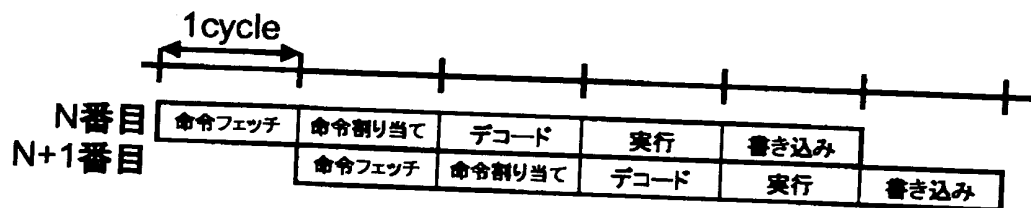


【図 17】

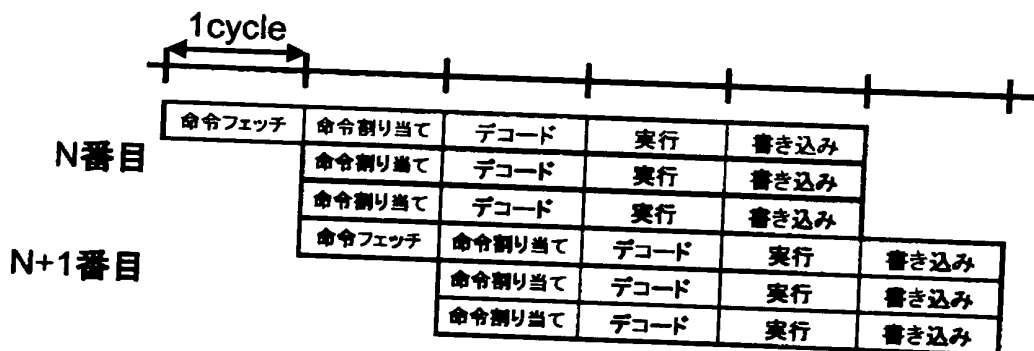




【図 18】

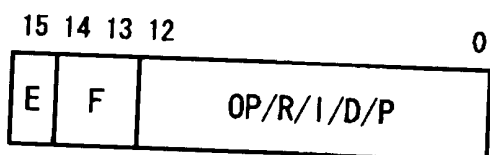


【図 19】

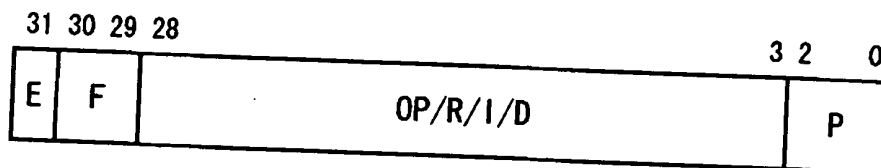


【図 20】

(a) 16ビット命令フォーマット



(b) 32ビット命令フォーマット



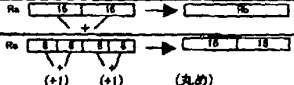
【図21】

カテゴリ	SIMD	サイズ	命令	オペランド	CFR	PSR	代表的な動作	演算器	
ALU add系	S I M D	ワード	add	Rc,Ra,Rb Rb,Ra,i12s SP,i19s Ra2,Rb2 Rc3,Ra3,Rb3 Ra2,i05s SP,i11s				31 16	
			addu	Rb,GP,i16u Rb,SP,i16u Ra3,SP,i08u				32 16	
			addc	Rc,Ra,Rb	W,caa,c0c1		キャリー付き加算	16	
			addvw	Rc,Ra,Rb	W,ovs		オーバーフローあり加算	32	
			sdds	Rc,Ra,Rb				16	
			addsr	Rc,Ra,Rb			$Ra + Rb \rightarrow Rb \ggg 1$	32	
			s1add	Rc,Ra,Rb Rc3,Ra3,Rb3			$Ra + Rb + 1 \rightarrow Rb \ggg 1$	16 32	
			s2add	Rc,Ra,Rb Rc3,Ra3,Rb3			$Ra + Rb \ggg 1 \rightarrow Rb \ggg 2$	16 32	
			addmsk	Rc,Ra,Rb	R-BP0			18	
			addarvw	Rc,Ra,Rb			$Ra \ggg 12 \text{ RBP } 1 \rightarrow Rb$	32	
	ハーフワード	fsddvh	Rc,Ra,Rb	W,ovs					
		vsddh	Rc,Ra,Rb	W,ovs					
		vsddvh	Rc,Ra,Rb	W,ovs		$Ra \text{ (16)} + Rb \text{ (16)} \rightarrow Rb \text{ (16)}$			
		vsaddh	Rb,Ra,i08s			$Rb \text{ (16)} + \text{即値} \rightarrow Rb \text{ (16)}$			
		vaddsh	Rc,Ra,Rb			$Ra \text{ (16)} + Rb \text{ (16)} \rightarrow Rb \text{ (16)}$			
		vsddsrh	Rc,Ra,Rb			$Ra \text{ (16)} + Rb \text{ (16)} \ggg 1 \rightarrow Rb \text{ (16)}$ (+1) (+1) (丸めあり)			
S I M D	ハーフワード	vaddhvc	Rc,Ra,Rb	RVC					
		vsddrhvc	Rc,Ra,Rb						
		vxaddh	Rc,Ra,Rb						
		vxaddvh	Rc,Ra,Rb	W,ovs		$Ra \text{ (16)} + Rb \text{ (16)} \rightarrow Rb \text{ (16)}$			
		vhaddh	Rc,Ra,Rb						
		vhaddvh	Rc,Ra,Rb	W,ovs		$Ra \text{ (16)} + Rb \text{ (16)} \rightarrow Rb \text{ (16)}$			
		viedh	Rc,Ra,Rb						
		viaddvh	Rc,Ra,Rb	W,ovs		$Ra \text{ (16)} + Rb \text{ (16)} \rightarrow Rb \text{ (16)}$			
		バイト	vsddb	Rc,Ra,Rb					
			vsaddb	Rb,Ra,i08s			$Rb \text{ (8)} + \text{即値} \rightarrow Rb \text{ (8)}$		
vaddsb	Rc,Ra,Rb								
vaddrb	Rc,Ra,Rb				$Ra \text{ (8)} + Rb \text{ (8)} \ggg 1 \rightarrow Rb \text{ (8)}$ (+1)(+1)(+1)(+1) (丸めあり)				

【図 2 2】

カテゴリ	SIMD	サイズ	命令	オペランド	CFR	PSR	代表的な動作	演算器	31 16 32 16
ALU sub 系	S I N G L E	ワード	sub	Rc,Rb,Ra Rb2,Ra2 Rc3,Rb3,Ra3			代表的な動作		31 16 32 16
			rsub	Rb,Ra,i08s Ra2,Rb2 Ra2,i04s			即値 - Ra → Rb (Rb2)		
			subc	Rc,Rb,Ra	W.cas.c0.c1		キャリー付き		
			subvw	Rc,Rb,Ra	W.ovs		オーバーフローあり		
			subs	Rc,Rb,Ra			Ra - Rb → Rb Rb - Rb → Rb		
			submsk	Rc,Rb,Ra	R.BP0		Ra - Rb → Rb Rb - Rb → Rb		
			ハーフワード	fsubvh	Rc,Rb,Ra		Ra - Rb → Rb Rb - Rb → Rb		
			vsubh	Rc,Rb,Ra			Ra - Rb → Rb Rb - Rb → Rb		
			vsubhvh	Rc,Rb,Ra	W.ovs		Ra - Rb → Rb Rb - Rb → Rb		
			varsubh	Rb,Ra,i08s			即値 - 即値 Rb - Rb → Rb		
	S I M D	ハーフワード	vsubsh	Rc,Rb,Ra			Ra - Rb → Rb Rb - Rb → Rb		
			vxsubh	Rc,Rb,Ra			Ra - Rb → Rb Rb - Rb → Rb		
			vxsubhvh	Rc,Rb,Ra	W.ovs		Ra - Rb → Rb Rb - Rb → Rb		
			vhsubh	Rc,Rb,Ra			Ra - Rb → Rb Rb - Rb → Rb		
			vhsubhvh	Rc,Rb,Ra	W.ovs		Ra - Rb → Rb Rb - Rb → Rb		
			vsubh	Rc,Rb,Ra			Ra - Rb → Rb Rb - Rb → Rb		
			vsubhvh	Rc,Rb,Ra	W.ovs		Ra - Rb → Rb Rb - Rb → Rb		
			vsubb	Rc,Rb,Ra			(即値) Rb - Rb → Rb		
			varsubb	Rb,Ra,i08s			Rb - Rb → Rb Rb - Rb → Rb		
			vasubb	Rc,Rb,Ra	RVC		Rb - Rb → Rb Rb - Rb → Rb		

【図23】

カテゴリ	SIMD	サイズ	命令	オペランド	CFR	PSR	代表的な動作	演算器	31 16
ALU logic 系	SINGLE	ワード	and	Rc,Ra,Rb Rb,Ra,i08u Ra2,Rb2			論理積	A	32
			andn	Rc,Ra,Rb Rb,Ra,i08u Ra2,Rb2					16
			or	Rc,Ra,Rb Rb,Ra,i08u Ra2,Rb2			論理和		32
			xor	Rc,Ra,Rb Rb,Ra,i08u Ra2,Rb2			排他的論理和		16
									32
ALU mov 系	SINGLE	ワード	mov	Rb,Reg32 Reg32,Rb Rb2,Reg16 Reg16,Rb2 Ra2,Rb Ra,i16s Ra2,i08s			Reg32 = TAR LR SVR PSR CFR MH0 MH1 ML0 ML1 EPSR IPC IPSR PC EPC PSR0 PSR1 PSR2 PSR3 CFR0 CFR1 CFR2 CFR3 Reg16 = TAR LR MH0 MH1	A	32
			movp	Rc,Rc+1,Ra,Rb			Rc ← Ra; Ra ← Rc		16
			movcf	Ck,Gi,Cm,Cn			Ck ← Gi; Cm ← Cn		32
			mvcdovs	Cm,Cm+1	W:ovs		Cm,Cm+1 ← CFR.OVS; CFR.OVS ← 0;		16
			mvdcas	Cm,Cm+1	W:cas		Cm,Cm+1 ← CFR.CAS; CFR.CAS ← 0;		32
			seth	Ra,i16s					16
									32
ALU max min 系	SINGLE	ワード	max	Rc,Ra,Rb	W:c0.c1		Rc ← max(Ra,Rb)	A	32
			min	Rc,Ra,Rb	W:c0.c1		Rc ← min(Ra,Rb)		16
			vmaxh	Rc,Ra,Rb					32
			vminh	Rc,Ra,Rb					16
			vmaxb	Rc,Ra,Rb					32
ALU abs 系	SINGLE	ワード	abs	Rb,Ra			絶対値	A	32
			abshw	Rb,Ra	W:ovs		オーバーフローあり		16
			abshv	Rb,Ra	W:ovs				32
			vabshw	Rb,Ra	W:ovs				16
			vabshv	Rb,Ra	W:ovs				32
ALU neg 系	SINGLE	ワード	negv	Rb,Ra	W:ovs			A	32
			fnegv	Rb,Ra	W:ovs				16
			vnegv	Rb,Ra	W:ovs				32
									16
									32
ALU sum 系	SINGLE	ワード	vsunh	Rb,Ra				A	32
			vsunh2	Rb,Ra					16
			vsunrh2	Rb,Ra					32
			vabsumb	Rc,Ra,Rb					C
									32
ALU その他	SINGLE	ワード	fmdvh	Rb,Ra	W:ovs		丸め	C	32
			vfmdvh	Rb,Mn	W:ovs				16
			vsel	Rc,Ra,Rb	RVC				32
			vsgh	Rb,Ra					C
									32

【図 24】

カテゴリ	SIMD	サイズ	命令	オペランド	CFR	PSR	代表的な動作	演算器	3116										
CMP	SINGLE		cmpCCn	Cm, Ra, Rb, Cn Cm, Ra, i05s, Cn Cm: Cmm+1, Ra, Rb, Cn Cm: Cmm+1, Ra, i05s, Cn	W:CF		CC = eq, ne, gt, ge, gtu, geu, le, lt, leu, leu Cm <- result & Cn; (Cmm+1 <- result & Cn)	A	32										
			cmpCCe	Cm: Cmm+1, Ra, Rb, Cn Cm: Cmm+1, Ra, i05s, Cn	W:CF		Cm <- result & Cn; Cmm+1 <- result & Cn;			18									
			cmpCCo	Cm: Cmm+1, Ra, Rb, Cn Cm: Cmm+1, Ra, i05s, Cn	W:CF		Cm <- result   Cn; Cmm+1 <- result   Cn;				32								
			cmpCC	C6, Ra2, Rb2 C6, Ra2, i04s	W:CF		CC = eq, ne, gt, ge, le, lt C6 <- result					18							
			testzn	Cm, Ra, Rb, Cn Cm, Ra, i05u, Cn Cm: Cmm+1, Ra, Rb, Cn Cm: Cmm+1, Ra, i05u, Cn	W:CF		Cm <- (Ra & Rb == 0) & Cn; (Cmm+1 <- (Ra & Rb == 0) & Cn)						32						
			testze	Cm: Cmm+1, Ra, Rb, Cn Cm: Cmm+1, Ra, i05u, Cn	W:CF		Cm <- (Ra & Rb == 0) & Cn; Cmm+1 <- ((Ra & Rb == 0) & Cn);							18					
			testzo	Cm: Cmm+1, Ra, Rb, Cn Cm: Cmm+1, Ra, i05u, Cn	W:CF		Cm <- (Ra & Rb == 0)   Cn; Cmm+1 <- ((Ra & Rb == 0)   Cn);								32				
			testzn	Cm, Ra, Rb, Cn Cm, Ra, i05u, Cn Cm: Cmm+1, Ra, Rb, Cn Cm: Cmm+1, Ra, i05u, Cn	W:CF		Cm <- (Ra & Rb != 0) & Cn; (Cmm+1 <- (Ra & Rb != 0) & Cn)									18			
			testne	Cm: Cmm+1, Ra, Rb, Cn Cm: Cmm+1, Ra, i05u, Cn	W:CF		Cm <- (Ra & Rb != 0) & Cn; Cmm+1 <- ((Ra & Rb != 0) & Cn);										32		
			testno	Cm: Cmm+1, Ra, Rb, Cn Cm: Cmm+1, Ra, i05u, Cn	W:CF		Cm <- (Ra & Rb != 0)   Cn; Cmm+1 <- ((Ra & Rb != 0)   Cn);											18	
			testz	C6, Ra2, Rb2 C6, Ra2, i04u	W:CF		C6 <- (Ra2 & Rb2 == 0)												32
			testn	C6, Ra2, Rb2 C6, Ra2, i04u	W:CF		C6 <- (Ra2 & Rb2 != 0)												
SIMD	ハーフワード	vcmpCCh	Ra, Rb vscmpCCh	W:CF		CC = eq, ne, gt, le, ge, lt	32												
		vcmpCCb	Ra, Rb vscmpCCb	W:CF		CC = eq, ne, gt, le, ge, lt													

【図 25】

カテゴリ	SIMD	サイズ	命令	オペランド	CFR	PSR	代表的な動作	演算器	31 16
mul 系	S I N G L E	ワード x ワード	mul	Mm,Rc,Ra,Rb Mm,Rb,Ra,i08s			<div> <math display="block">\begin{array}{c} Rb \\ \times \\ Ra \\ \hline \end{array}</math>           (即値)         </div>	X2	
			mulu	Mm,Rc,Ra,Rb Mm,Rb,Ra,i08s			符号なし乗算		
			fmulhww	Mm,Rc,Ra,Rb		fxp	固定小数点演算		
		ワード x ハーフワード	hmul	Mm,Rc,Ra,Rb			<div> <math display="block">\begin{array}{c} Rb \\ \times \\ Ra \\ \hline \end{array}</math>           (16 16)         </div>	X1	
			imul	Mm,Rc,Ra,Rb			<div> <math display="block">\begin{array}{c} Rb \\ \times \\ Ra \\ \hline \end{array}</math>           (16 16)         </div>		
			fmulhww	Mm,Rc,Ra,Rb		fxp	<div> <math display="block">\begin{array}{c} Rb \\ \times \\ Ra \\ \hline \end{array}</math>           (16 16)         </div>		
		ハーフワード x ハーフワード	fmulhw	Mm,Rc,Ra,Rb		fxp	<div> <math display="block">\begin{array}{c} Rb \\ \times \\ Ra \\ \hline \end{array}</math>           (16 16)         </div>	X1	
			fmulhh	Mm,Rc,Ra,Rb		fxp	<div> <math display="block">\begin{array}{c} Rb \\ \times \\ Ra \\ \hline \end{array}</math>           (16 16)         </div>		
			fmulhhr	Mm,Rc,Ra,Rb		fxp	<div> <math display="block">\begin{array}{c} Rb \\ \times \\ Ra \\ \hline \end{array}</math>           (16 16)         </div>		
	S I M D	ハーフワード x ハーフワード	vmul	Mm,Rc,Ra,Rb			丸めあり	X2	32
			vfmulw	Mm,Rc,Ra,Rb		fxp	<div> <math display="block">\begin{array}{c} Rb \\ \times \\ Ra \\ \hline \end{array}</math>           (16 16)         </div>		
			vfmulh	Mm,Rc,Ra,Rb		fxp	<div> <math display="block">\begin{array}{c} Rb \\ \times \\ Ra \\ \hline \end{array}</math>           (16 16)         </div>		
			vfmulhr	Mm,Rc,Ra,Rb		fxp	<div> <math display="block">\begin{array}{c} Rb \\ \times \\ Ra \\ \hline \end{array}</math>           (16 16)         </div>		
			vzmul	Mm,Rc,Ra,Rb			丸めあり		
			vzfmulw	Mm,Rc,Ra,Rb		fxp	<div> <math display="block">\begin{array}{c} Rb \\ \times \\ Ra \\ \hline \end{array}</math>           (16 16)         </div>		
			vzfmulh	Mm,Rc,Ra,Rb		fxp	<div> <math display="block">\begin{array}{c} Rb \\ \times \\ Ra \\ \hline \end{array}</math>           (16 16)         </div>		
			vzfmulhr	Mm,Rc,Ra,Rb		fxp	<div> <math display="block">\begin{array}{c} Rb \\ \times \\ Ra \\ \hline \end{array}</math>           (16 16)         </div>		
			vfmul	Mm,Rc,Ra,Rb		fxp	丸めあり		
			vzfmulw	Mm,Rc,Ra,Rb		fxp	<div> <math display="block">\begin{array}{c} Rb \\ \times \\ Ra \\ \hline \end{array}</math>           (16 16)         </div>		
			vzfmulh	Mm,Rc,Ra,Rb		fxp	<div> <math display="block">\begin{array}{c} Rb \\ \times \\ Ra \\ \hline \end{array}</math>           (16 16)         </div>		
			vzfmulhr	Mm,Rc,Ra,Rb		fxp	<div> <math display="block">\begin{array}{c} Rb \\ \times \\ Ra \\ \hline \end{array}</math>           (16 16)         </div>		
			vfmul	Mm,Rc,Ra,Rb		fxp	丸めあり		
			vzfmulw	Mm,Rc,Ra,Rb		fxp	<div> <math display="block">\begin{array}{c} Rb \\ \times \\ Ra \\ \hline \end{array}</math>           (16 16)         </div>		
			vzfmulh	Mm,Rc,Ra,Rb		fxp	<div> <math display="block">\begin{array}{c} Rb \\ \times \\ Ra \\ \hline \end{array}</math>           (16 16)         </div>		
			vzfmulhr	Mm,Rc,Ra,Rb		fxp	<div> <math display="block">\begin{array}{c} Rb \\ \times \\ Ra \\ \hline \end{array}</math>           (16 16)         </div>		
		ワード x ハーフワード	vzfmulhww	Mm,Rc,Rc+1,Ra,Rb Mm,Rc,Rc+1,Ra,Rb		fxp	丸めあり		

【図26】

カテゴリ	SIMD	サイズ	命令	オペランド	CFR	PSR	代表的な動作	演算部	31 18
mac系	SINGLE	ワード×ワード	mac	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx			mulを使った積和演算	X2	32
			fmacww	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx		fxp	fmulwwを使った積和演算		
		ワード ×ハーフワード	hmac	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx			hmulを使った積和演算	X1	
			lmac	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx			lmulを使った積和演算		
			fmachww	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx		fxp	fmulhwwを使った積和演算		
			fmachw	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx		fxp	fmulhwを使った積和演算		
			fmachh	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx		fxp	fmulhhを使った積和演算		
			fmachhr	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx		fxp	丸めあり		
		ハーフワード ×ハーフワード	vmac	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx			vmulを使った積和演算	X2	
			vfmacw	Mm,Rc,Ra,Rb,Mn		fxp	vfmulwを使った積和演算		
			vrmac	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx			vrmulを使った積和演算		
			vrfmacw	Mm,Rc,Ra,Rb,Mn		fxp	vrfmulwを使った積和演算		
	vxfmach		Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx		fxp	vxfmulhを使った積和演算			
	vxfmachr		Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx		fxp	丸めあり			
	vhnac		Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx			vhnulを使った積和演算			
	vhnacw		Mm,Rc,Ra,Rb,Mn		fxp	vhnulwを使った積和演算			
	vhnfmach		Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx		fxp	vhnfmulhを使った積和演算			
	vhnfmachr		Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx		fxp	丸めあり			
	SIMD	ハーフワード ×ハーフワード	vlmac	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx			vlmulを使った積和演算	X2	
			vlfmacw	Mm,Rc,Ra,Rb,Mn		fxp	vlfmulwを使った積和演算		
			vfmach	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx		fxp	vfmulhを使った積和演算		
			vfmachr	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx		fxp	丸めあり		
			vlfmach	Mm,Rc,Ra,Rb,Mn Mm,Rc,Ra,Rb,Mn		fxp	vlfmulhを使った積和演算		
			vlfmachr	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx		fxp	丸めあり		
			ワード× ハーフワード	vpfmachww	Mm,Rc,Rc+1,Ra,Rb,Mn		fxp		

【図 27】

カテゴリ	SDMO	サイズ	命令	オペランド	CFR	PSR	代表的な動作	演算器	31 16
msu系	S I N G L E	ワード x ワード	msu	Mm, Rc, Ra, Rb, Mn M0, Rc, Ra, Rb, Rx			mulを使った積差演算	X2	32
			fmsuww	Mm, Rc, Ra, Rb, Mn M0, Rc, Ra, Rb, Rx	fxp	fmsuwwを使った積差演算			
		ワード x ハーフワード	hmsu	Mm, Rc, Ra, Rb, Mn M0, Rc, Ra, Rb, Rx			hmulを使った積差演算	X1	
			hmsu	Mm, Rc, Ra, Rb, Mn M0, Rc, Ra, Rb, Rx			hmulを使った積差演算		
			fmsuhw	Mm, Rc, Ra, Rb, Mn M0, Rc, Ra, Rb, Rx	fxp	fmsuhwを使った積差演算			
			fmsuhw	Mm, Rc, Ra, Rb, Mn M0, Rc, Ra, Rb, Rx	fxp	fmsuhwを使った積差演算			
		ハーフワード x ハーフワード	fmsuhh	Mm, Rc, Ra, Rb, Mn M0, Rc, Ra, Rb, Rx	fxp	fmsuhhを使った積差演算	丸めあり		
			fmsuhh	Mm, Rc, Ra, Rb, Mn M0, Rc, Ra, Rb, Rx	fxp	fmsuhhを使った積差演算			
			fmsuhh	Mm, Rc, Ra, Rb, Mn M0, Rc, Ra, Rb, Rx	fxp	丸めあり			
			fmsuhh	Mm, Rc, Ra, Rb, Mn M0, Rc, Ra, Rb, Rx	fxp	丸めあり			
		ハーフワード x ハーフワード	vmsu	Mm, Rc, Ra, Rb, Mn M0, Rc, Ra, Rb, Rx			vmsuを使った積差演算	X2	
			vmsuw	Mm, Rc, Ra, Rb, Mn	fxp	vmsuwを使った積差演算			
	vmsuh		Mm, Rc, Ra, Rb, Mn M0, Rc, Ra, Rb, Rx	fxp	vmsuhを使った積差演算				
	vmsu		Mm, Rc, Ra, Rb, Mn M0, Rc, Ra, Rb, Rx			vmsuを使った積差演算			
	vxfmsuw		Mm, Rc, Ra, Rb, Mn	fxp	vxfmsuwを使った積差演算				
	vxfmsuh		Mm, Rc, Ra, Rb, Mn M0, Rc, Ra, Rb, Rx	fxp	vxfmsuhを使った積差演算				
	vhmsu		Mm, Rc, Ra, Rb, Mn M0, Rc, Ra, Rb, Rx			vhmsuを使った積差演算			
	vhmsuw		Mm, Rc, Ra, Rb, Mn	fxp	vhmsuwを使った積差演算				
	vhfmsuh		Mm, Rc, Ra, Rb, Mn M0, Rc, Ra, Rb, Rx	fxp	vhfmsuhを使った積差演算				
	ハーフワード x ハーフワード	vhfmsu	Mm, Rc, Ra, Rb, Mn M0, Rc, Ra, Rb, Rx	fxp	vhfmsuを使った積差演算				
		vffmsu	Mm, Rc, Ra, Rb, Mn M0, Rc, Ra, Rb, Rx			vffmsuを使った積差演算			
vffmsuw		Mm, Rc, Ra, Rb, Mn	fxp	vffmsuwを使った積差演算					
vffmsuh		Mm, Rc, Ra, Rb, Mn M0, Rc, Ra, Rb, Rx	fxp	vffmsuhを使った積差演算					
vffmsu		Mm, Rc, Ra, Rb, Mn M0, Rc, Ra, Rb, Rx			vffmsuを使った積差演算				
vffmsuw		Mm, Rc, Ra, Rb, Mn	fxp	vffmsuwを使った積差演算					
vffmsuh		Mm, Rc, Ra, Rb, Mn M0, Rc, Ra, Rb, Rx	fxp	vffmsuhを使った積差演算					



【図28】

カテゴリ	SIMD	サイズ	命令	オペランド	CFR	PSR	代表的な動作	演算器	31
MEM Id 系	シ ン グ ル	ワード	ld	Rb.(Ra,d10u) Rb.(GP,d13u) Rb.(SP,d13u) Rb.(Ra+),10s Rb2.(Ra2) Rb2.(Ra2,d05u) Rb2.(GP,d06u) Rb2.(SP,d06u) Rb2.(Ra2+)			レジスタ ← メモリ 32 ← 32		32
			ldh	Rb.(Ra,d09u) Rb.(GP,d12u) Rb.(SP,d12u) Rb.(Ra+),09s Rb2.(Ra2) Rb3.(Ra3,d04u) Rb2.(GP,d05u) Rb2.(SP,d05u) Rb2.(Ra2+)			32 ← 16		16
			ldhu	Rb.(Ra,d09u) Rb.(GP,d12u) Rb.(SP,d12u) Rb.(Ra+),09s					32
		バイト	ldb	Rb.(Ra,d08u) Rb.(GP,d11u) Rb.(SP,d11u) Rb.(Ra+),08s			レジスタ ← メモリ 32 ← 8		16
			ldbu	Rb.(Ra,d08u) Rb.(GP,d11u) Rb.(SP,d11u) Rb.(Ra+),08s					32
		バイト→ ハーフワード	ldbh	Rb.(Ra+),07s			16 ← 8 8		
			ldbuh	Rb.(Ra+),07s					
		ワード	ldp	Rb:Rb+1.(Ra,d11u) LR:SVR.(Ra,d11u) TAR:UDR.(Ra,d11u) Rb:Rb+1.(GP,d14u) LR:SVR.(GP,d14u) TAR:UDR.(GP,d14u) Rb:Rb+1.(SP,d14u) LR:SVR.(SP,d14u) TAR:UDR.(SP,d14u) Rb:Rb+1.(Ra+),11s Rb:Rb+1.(SP,d07u) LR:SVR.(SP,d07u) Rb2:Re2.(Ra2+)			32 ← 32 32 ← 32		
			ldhp	Rb:Rb+1.(Ra,d10u) Rb:Rb+1.(Ra+),10s Rb2:Re2.(Ra2+)			32 ← 16 16		16
			ldbp	Rb:Rb+1.(Ra,d09u) Rb:Rb+1.(Ra+),09s			32 ← 8 8 8		32
		バイト→ ハーフワード	ldbhp	Rb:Rb+1.(Ra+),07s			16 ← 8 8 8 8		16
		バイト→ ハーフワード	ldbuhp	Rb:Rb+1.(Ra+),07s			16 ← 8 8 8 8		32

【図 29】

カテゴリー	SMD	サイズ	命令	オペランド	CFR	PSR	代表的な動作	演算器	31
MEM store 系	SINGL E	ワード	st	(Ra,d10u),Rb (GP,d13u),Rb (SP,d13u),Rb (Ra+),Rb (Ra2),Rb2 (Ra2,d05u),Rb2 (GP,d06u),Rb2 (SP,d06u),Rb2 (Ra2+),Rb2			レジスタ 32 → メモリ 32		32
			sth	(Ra,d09u),Rb (GP,d12u),Rb (SP,d12u),Rb (Ra+),Rb (Ra2),Rb2 (Ra2,d04u),Rb2 (GP,d05u),Rb2 (SP,d05u),Rb2 (Ra2+),Rb2			16 → 16		16
		バイト	stb	(Ra,d08u),Rb (GP,d11u),Rb (SP,d11u),Rb (Ra+),Rb (Ra2),Rb2			8 → 8		32
		バイト→ ハーフワード	stbh	(Ra+),Rb			8 8 → 16		16
		ワード	stp	(Ra,d11u),Rb,Rb+1 (Ra,d11u),LR:SVR (Ra,d11u),TAR:UDR (GP,d14u),Rb,Rb+1 (GP,d14u),LR:SVR (GP,d14u),TAR:UDR (SP,d14u),Rb,Rb+1 (SP,d14u),LR:SVR (SP,d14u),TAR:UDR (Ra+),Rb,Rb+1 (SP,d07u),Rb,Rb (SP,d07u),LR:SVR (Ra2+),Rb2,Rb2			32 32 → 32 32		32
			sthp	(Ra,d10u),Rb,Rb+1 (Ra+),Rb,Rb+1 (Ra2+),Rb2,Rb2			16 16 → 32		16
			stbp	(Ra,d09u),Rb,Rb+1 (Ra+),Rb,Rb+1			8 8 → 16		32
			stbhp	(Ra+),Rb,Rb+1			8 8 8 8 → 32		16
		ハーフワード	sthp	(Ra,d09u),Rb,Rb+1 (Ra+),Rb,Rb+1			8 8 → 16		32
			stbhp	(Ra+),Rb,Rb+1			8 8 8 8 → 32		32

【図30】

カテゴリ	SIMD	サイズ	命令	オペランド	CFR	PSR	代表的な動作	演算 器	
BRA			setlr	d09s C5.d09s			LRを設定	B	31 16
			settar	d09s C8.d09s C8.C2.C4.d09s C8.Cm.d09s C8.C4.d09s	W:c6 W:c2.c4.c6 W:c6.cm W:c6		LRからフェッチした命令を分岐用バッファに格納 TARを設定 TARからフェッチした命令を分岐用バッファに格納		32
			setbb	LR TAR			LRからフェッチした命令を分岐用バッファに格納 TARからフェッチした命令を分岐用バッファに格納		16
			jloop	C5.LR.Ra.i08s C8.TAR.Ra.i08s C8.C2.C4.TAR.Ra.i08s C8.Cm.TAR.Ra.i08s C8.TAR.Ra2 C8.C2.C4.TAR.Ra2 C8.Cm.TAR.Ra2	W:c5 W:c6 W:c2.c4.c6 W:c6.cm W:c6 W:c2.c4.c6 W:c6		ブレイクアウト[c5]のみ ブレイクアウト[c6]のみ		32
			jmp	TAR LR					16
			jmp1	TAR LR	R:CF				
			jmpf	TAR LR Cm.TAR C8.C2.C4.TAR					
			jmprr	LR					
			br	d20s d09s				32 16 32 16	
			br1	d20s d09s	R:CF		ブレイクアウト[c6][c7]のみ		
			rti			W:PSR R:eh			

【図31】

カテゴリ	SIMDサイズ	命令	オペランド	CFR	PSR	代表的な動作	演算器	31 16
BS asi系	SINGLE	ワード	asi	Rc, Ra, Rb Rb, Ra, i05u Ra2, i04u		左シフト ←Ra[04] Ra → Rc	S1	32 16
			faslvw	Rc, Ra, Rb Rb, Ra, i05u Rc, Ra, Rb Rb, Ra, i05u	W.ovs	←Ra[04] Ra 飽和あり → Rc		
		ペアワード	aslp	Mm, Ra, Mn, Rb Mm, Rb, Mn, i06u Mm, Rc, Mh, Ra, Rb Mm, Rb, Mh, Ra, i06u		←Ra[04] MhMn MhMn → MhMn MhMn Rb	S2	32
			faslpvw	Mm, Ra, Mn, Rb Mm, Rb, Mn, i06u	W.ovs	←Ra[04] MhMn MhMn 飽和あり → MhMn MhMn Rb		
	SIMD	ワード	vasl	Mm, Ra, Mn, Rb Mm, Rb, Mn, i05u		MhMn MhMn (1) (2) (1) (2) → (1) (2) (1) (2) (1)×Ra[04] (2)×Ra[04]	S2	32
			vfaslvw	Mm, Ra, Mn, Rb Mm, Rb, Mn, i05u	W.ovs	MhMn MhMn 飽和あり → MhMn MhMn (1) (2) (1) (2) (1) (2) (1)×Ra[04] (2)×Ra[04]		
		ハーフワード	vaslh	Rc, Ra, Rb Rb, Ra, i04u		←Ra[03] ←Ra[03] Ra (1) (2) → (1) (2) Rc	S1	32
			vfaslvh	Rc, Ra, Rb Rb, Ra, i04u	W.ovs	←Ra[03] ←Ra[03] Ra (1) (2) 飽和あり → (1) (2) Rc		
		バイト	vaslb	Rc, Ra, Rb Rb, Ra, i03u		←Ra[02] ←Ra[02] Ra (1) (2) (3) (4) → (1) (2) (3) (4) Rc		
	BS asr系	ワード	asr	Rc, Ra, Rb Rb, Ra, i05u Ra2, i04u		算術右シフト →Ra[04] Ra → Rc	S1	32 16
			asrp	Mm, Ra, Mn, Rb Mm, Rb, Mn, i06u Mm, Rc, Mh, Ra, Rb Mm, Rb, Mh, Ra, i06u		→Ra[04] MhMn MhMn → MhMn MhMn Rb		
		ワード	vasr	Mm, Ra, Mn, Rb Mm, Rb, Mn, i05u		MhMn MhMn (1) (2) (1) (2) → (1) (2) (1) (2) (1)×Ra[04] (2)×Ra[04]	S2	32
		ハーフワード	vasrh	Rc, Ra, Rb Rb, Ra, i04u		→Ra[03] →Ra[03] Ra (1) (2) → (1) (2) Rc		
		バイト	vasrb	Rc, Ra, Rb Rb, Ra, i03u		→Ra[02] →Ra[02] Ra (1) (2) (3) (4) → (1) (2) (3) (4) Rc	S1	

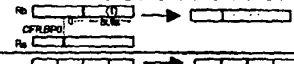
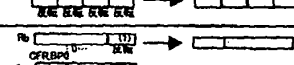
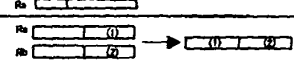
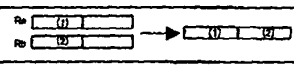
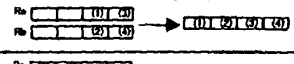
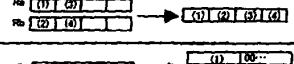
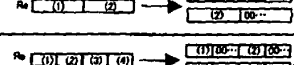
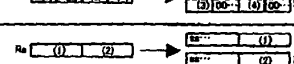
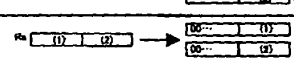
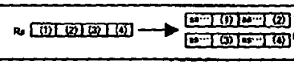
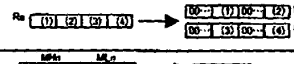
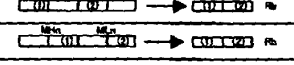
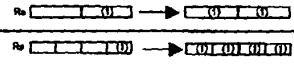
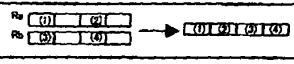




【図32】

カテゴリ	SIMDサイズ	命令	オペランド	CFR	PSR	代表的な動作	演算器	31 16
BS lar系	SINGLE	ワード	lar Rc,Ra,Rb Rb,Ra,i06u			論理右シフト $\ggg Ra(04)$ 	S1	32
		ベアワード	larp Mm,Ra,Mn,Rb Mm,Rb,Mn,i06u Mm,Rc,MHn,Ra,Rb Mm,Rb,MHn,Ra,i06u			$\ggg Ra(04)$ 	S2	
	SIMD	ワード	visr Mm,Ra,Mn,Rb Mm,Rb,Mn,i05u				S2	
		ハーフワード	visrh Rc,Ra,Rb Rb,Ra,i04u			$\ggg Ra(02) \ggg Ra(02)$ 	S1	
		バイト	visrb Rc,Ra,Rb Rb,Ra,i03u			$\ggg Ra(02) \ggg Ra(02)$ $\ggg Ra(02) \ggg Ra(02)$ 	S1	
BS rotate系	SINGLE	ワード	rol Rc,Ra,Rb Rb,Ra,i05u				S1	32
	SIMD	ハーフワード	vroh Rc,Ra,Rb Rb,Ra,i04u					
		バイト	vrob Rc,Ra,Rb Rb,Ra,i03u					
BS ext系	SINGLE	ワード	extw Mm,Rb,Ra				C	32
		ハーフワード	exth Ra2				S2	
	SIMD	ハーフワード	exthu Ra2				S2	16
		バイト	extb Ra2				S2	
	SIMD	ハーフワード	extbu Ra2				C	32

【図33】

カテゴリ	SIMDサイズ	命令	オペランド	CFR	PSR	代表的な動作	演算器	31 16
CNV valn系	SIMD	valn	Rc,Ra,Rb	R:aln(1:0)		$\ll (CFR:aln(1:0) \ll 3)$ $(a3:32)$ 	C	32
		valn1	Rc,Ra,Rb					
		valn2	Rc,Ra,Rb					
		valn3	Rc,Ra,Rb					
		valnvc1	Rc,Ra,Rb	R:VC0				
		valnvc2	Rc,Ra,Rb	R:VC0				
		valnvc3	Rc,Ra,Rb	R:VC0				
		valnvc4	Rc,Ra,Rb	R:VC0				

【図34】

カテゴリ	命令	オペランド	CFR	PSR	代表的な動作	演算器
CNV	S I N G L E	bcntl	Rb, Ra		1の数をカウント	C 32
		bseq0	Rb, Ra		MSBから最初に0になるまでをカウント	
		bseq1	Rb, Ra		MSBから最初に1になるまでをカウント	
		bseq	Rb, Ra		MSB-1から最初にMSBになるまでをカウント	
		makbrvh	Rc, Ra, Rb	R.BP0		
		byterev	Rb, Ra			
		maskbrvh	Rc, Ra, Rb	R.BP0		
	ハーフワード	vinthh	Rc, Ra, Rb			
		vinthh	Rc, Ra, Rb			
	バイト	vinthb	Rc, Ra, Rb			
		vinthb	Rc, Ra, Rb			
	ハーフワード	vhunplh	Rb:Rb+1, Ra			
		vhunplb	Rb:Rb+1, Ra			
	ハーフワード	vhunplh	Rb:Rb+1, Ra			
		vhunplhu	Rb:Rb+1, Ra			
	バイト	vhunplb	Rb:Rb+1, Ra			
		vhunplbu	Rb:Rb+1, Ra			
	ハーフワード	vhunpl1	Rb, Mn			
		vhunpl2	Rb, Mn			
	バイト	vstovh	Rb, Ra			
		vstovb	Rb, Ra			
		vhplb	Rc, Ra, Rb			

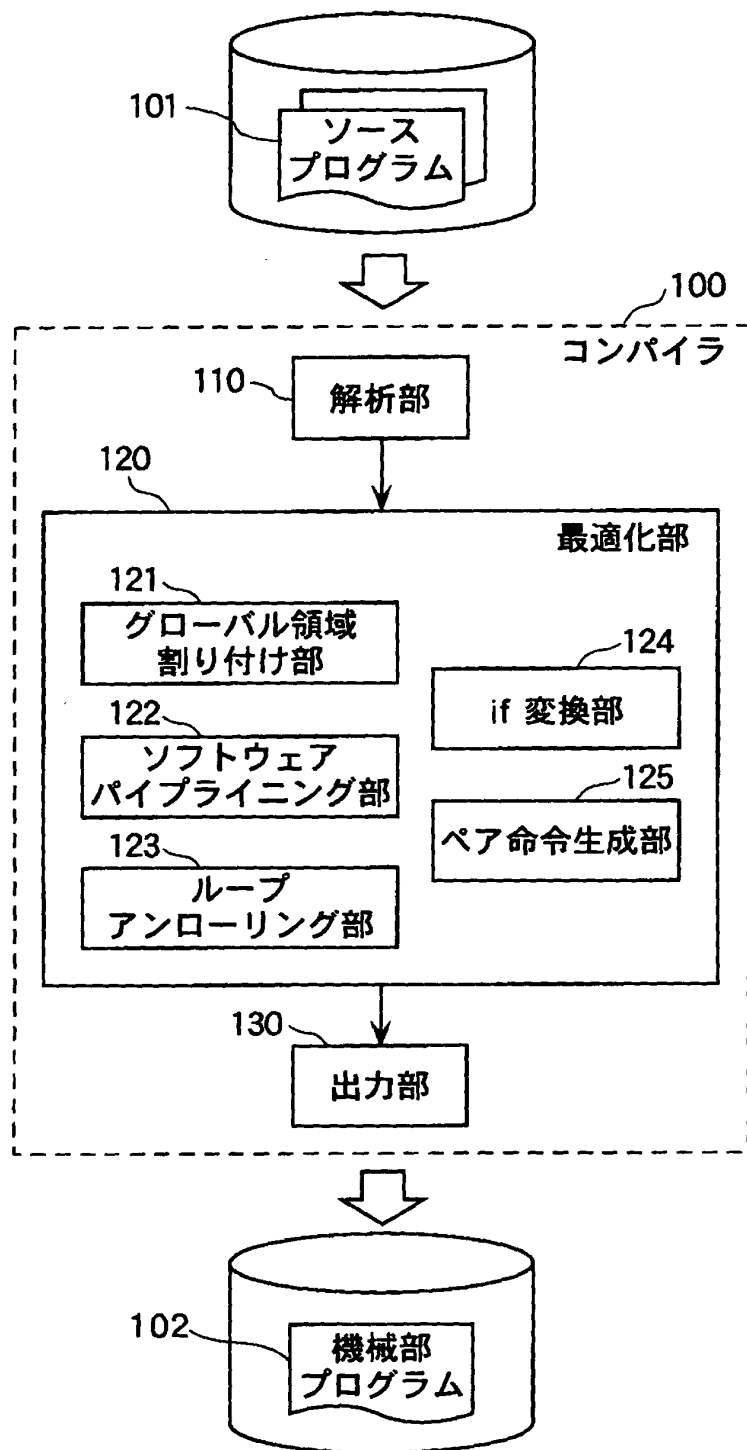
【図35】

カテゴリ	SIMD	サイズ	命令	オペランド	CFR	PSR	代表的な動作	演算器	31 16
SAT vblk 系	SIMD	ワード→ ハーフワード	vblkh	Rc,Ra,Rb				C	32
			vblkhu	Rc,Ra,Rb					
		ハーフワード→ バイト	vblkb	Rc,Ra,Rb					
			vblkbu	Rc,Ra,Rb					
SAT sat 系	SIMD	ワード	satw	Min,Rb,Min			ワード飽和	C	32
			sath	Rb,Ra			ハーフワード飽和		
			satb	Rb,Ra			バイト飽和		
			satbu	Rb,Ra			バイト符号なし飽和		
			sat9	Rb,Ra			9ビット飽和		
			sat12	Rb,Ra			12ビット飽和		
	SIMD	ハーフワード	vsath	Min,Rb,Min					
			vsath8	Rb,Ra			符号付き8ビット飽和		
			vsath8u	Rb,Ra			符号なし8ビット飽和		
			vsath9	Rb,Ra			9ビット飽和		
			vsath12	Rb,Ra			12ビット飽和		

【図36】

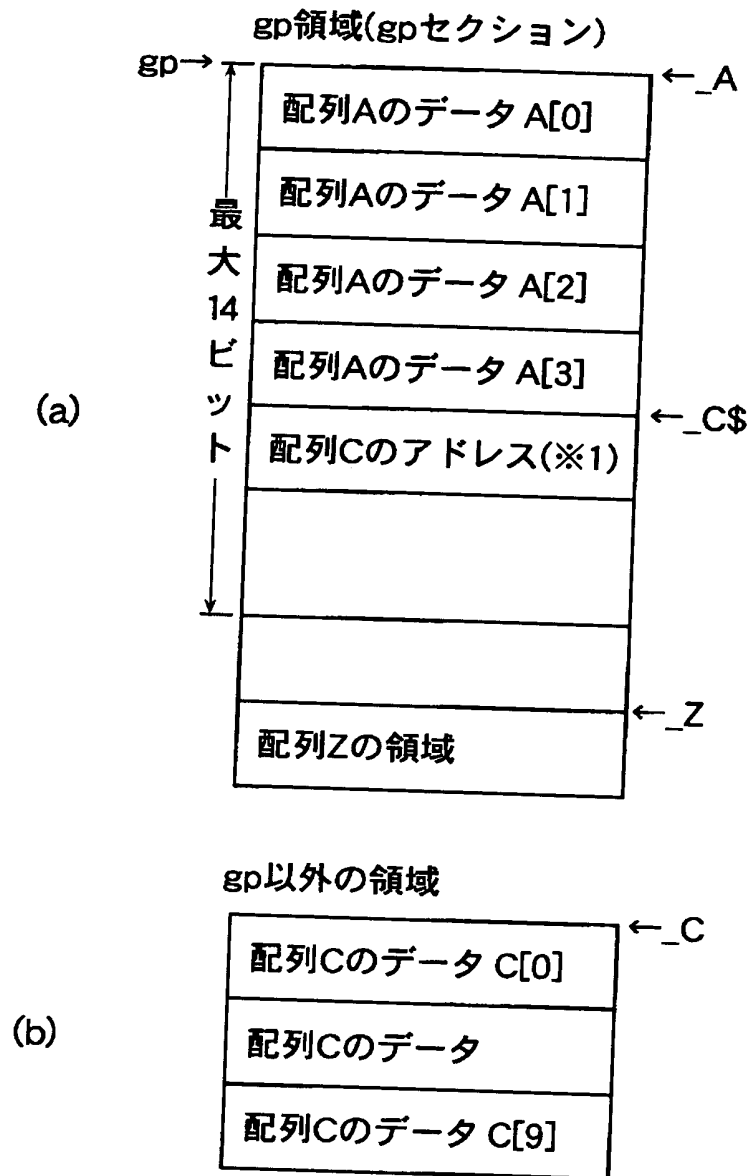
カテゴリ	SIMD	サイズ	命令	オペランド	CFR	PSR	代表的な動作	演算器	31 16
MSK			mskgen	Rc,Rb Rb,i05U,i05u			マスク生成 	S2	32
			msk	Rc,Ra,Rb Rb,Ra,i05U,i05u					
EXTR			extr	Rc,Ra,Rb Rb,Ra,i05U,i05u				S2	32
			extru	Rc,Ra,Rb Rb,Ra,i05U,i05u					
DIV			div	MHiM,Rc,MHiM,Ra,Rb	Wzvs		除算	DIV	32
			divu	MHiM,Rc,MHiM,Ra,Rb					
ETC			piNi			Wzhi,ie,pl R:PSR	ソフトウェア割り込みN=0~7	B	32
			piN			Wzhi,ie,pl R:PSR	ソフトウェア割り込みN=0~7		
			scN			Wzhi,ie,pl R:PSR	システムコールN=0~7		
			ldstb	Rb,(Ra)			load/バロック	M	32
			rd	Rb,(Ra) Rb,(d1 iu) Rb2,(Ra2)		R:ee	外部レジスタリード		
			wt	(Ra),Rb (d1 iu),Rb (Ra2),Rb2		R:ee	外部レジスタライト		
			dpref	(Ra,d1 iu)			プリフェッチ	DBGM	16
			dbgmn	i18u			N=0~3		
			vcchk		WCF RVC		VCフラグチェック		
			vmpsw				VMP切替え	B	32
			vmpsw	LR			VMP切替え禁止		
			vmpintd1			W:ie	VMP切替え許可		
			vmpintd2			W:ie	VMP切替え許可		
			vmpintd3			W:ie	VMP切替え許可		
			vmpinte1			W:ie	VMP切替え許可	A	16
			vmpinte2						
			vmpinte3						
			nop				no operation	A	16

【図 37】

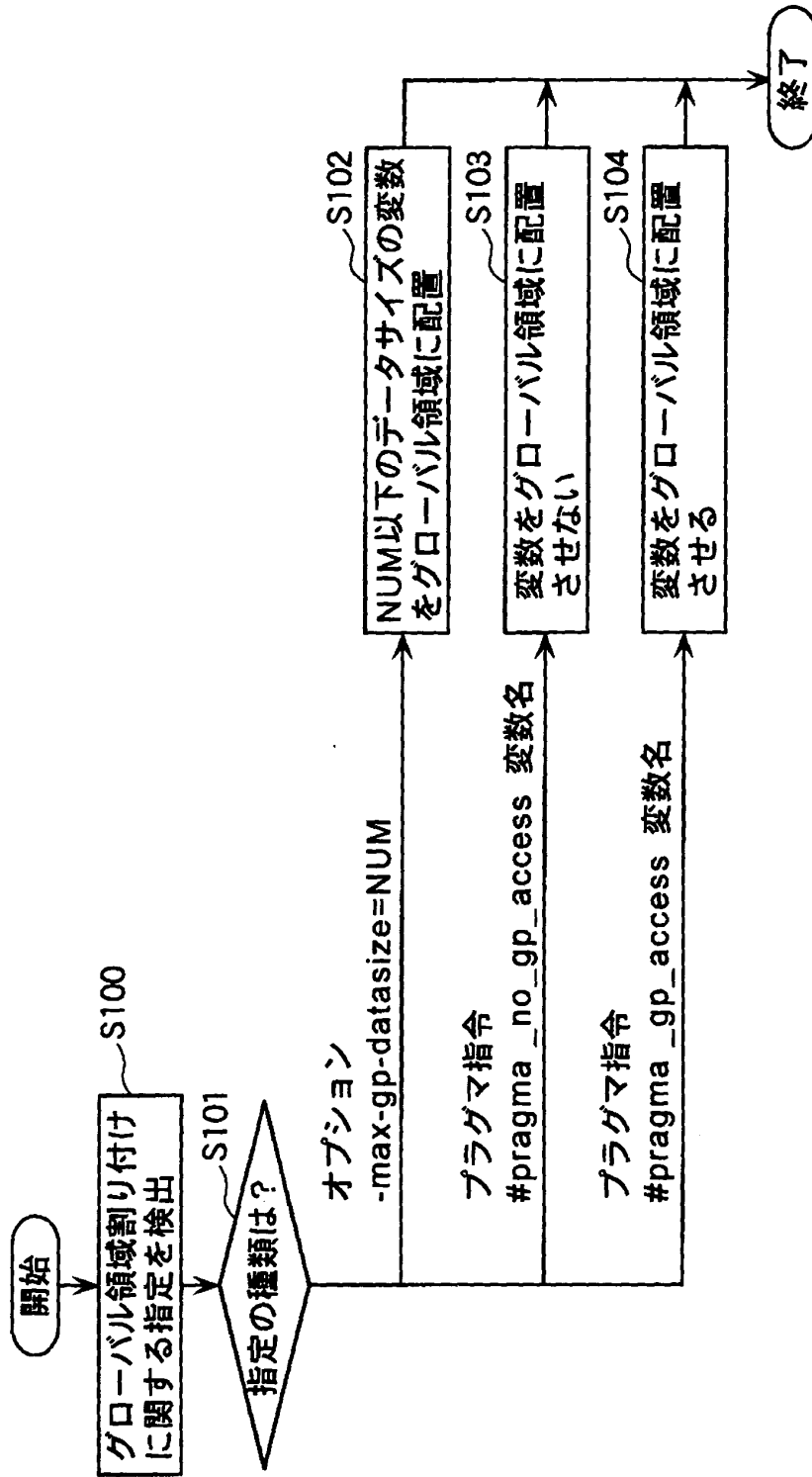




【図 3 8】



【図39】



【図 4 0】

最大データサイズの変化による生成コードの比較

デフォルト	-mmax-gp-datasize=40を指定
<pre> inta[8]; intc[10];  intsample(void) {   a[7]=c[0]+c[9];    return a[7]; } </pre>	<pre> inta[8]; intc[10];  intsample(void) {   a[7]=c[0]+c[9];    return a[7]; } </pre>
<pre> ld r1,(gp,_c\$-.MN.gptop);; setlor0,LO(_c+36);; sethir0,HI(_c+36);; ld r1,(r1);; ld r0,(r0);; add r0,r1,r0;; st(gp,_a-.MN.gptop+28),r0 ret;; </pre>	<pre> ld r1,(gp,_c-.MN.gptop);; ld r0,(gp,_c-.MN.gptop+36);;  add r0,r1,r0;; st(gp,_a-.MN.gptop+28),r0 ret;; </pre>
10サイクル 8バイト	7サイクル 5バイト

【図41】

gp領域配置の最大データサイズを40とした場合

サイズ指定なしファイル外定義	ファイル内定義/ サイズ指定ファイル外定義
<pre>extern int a[ ]; extern int c[ ];  int sample(void) {     a[7] = c[0] + c[9];      return a[7]; }</pre>	<pre>int a[8]; extern int c[10];  int sample(void) {     a[7] = c[0] + c[9];      return a[7]; }</pre>
<pre>setlo r0,LO(_c);; setlo r1,LO(_c+36) sethi r0,HI(_c);; sethi r1,HI(_c+36);; ld  r3,(r0);; setlo r2,LO(_a+28) ld  r0,(r1);; sethi r2,HI(_a+28);; add r1,r3,r0;; mov r0,r1 st  (r2),r1 ret  ;;</pre>	<pre>ld r1,(gp,_c - .MN.gptop);; ld r0,(gp,_c - .MN.gptop+36);;  add r0,r1,r0;;  st (gp,_a - .MN.gptop+28),r0 ret;;</pre>
10サイクル 12バイト	7サイクル 5バイト

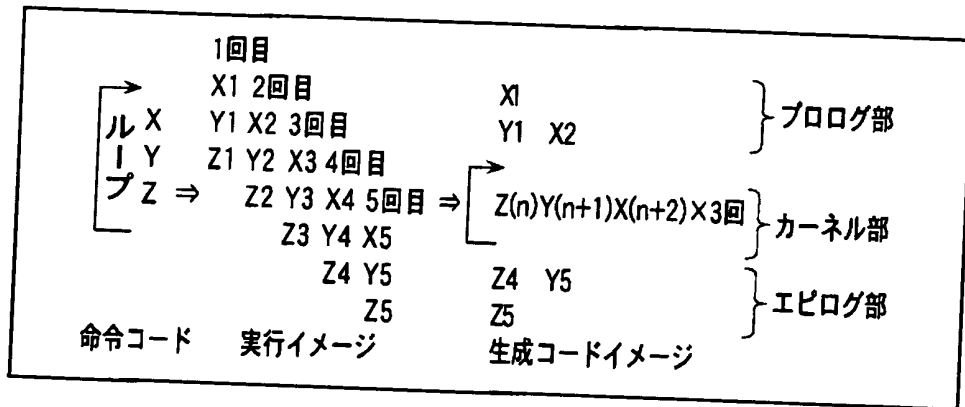
【図 4 2】

gp領域配置の最大データサイズをデフォルトの32とした場合

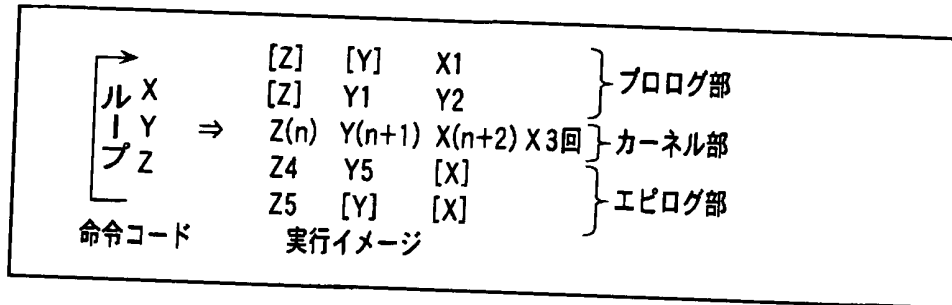
サイズ指定ありファイル外定義 / ファイル内定義 #pragma _no_gp_access指令	サイズ指定なしファイル外定義 / ファイル内定義 #pragma _gp_access指令
<pre>#pragma _no_gp_access a, c  extern int a[8]; int c[10];  int sample(void) {     a[7] = c[0] + c[9];     return a[7]; }</pre>	<pre>#pragma _gp_access a, c  extern int a[]; int c[10];  int sample(void) {     a[7] = c[0] + c[9];     return a[7]; }</pre>
<pre>setlo r0,LO(_c);; setlo r1,LO(_c+36) sethi r0,HI(_c);; sethi r1,HI(_c+36);; ld r3,(r0);; setlo r2,LO(_a+28) ld r0,(r1);; sethi r2,HI(_a+28);; add r1,r3,r0;; mov r0,r1 st (r2),r1 ret ;;</pre>	<pre>ld r1,(gp,_c - .MN.gptop);; ld r0,(gp,_c - .MN.gptop+36);; add r0,r1,r0;; st (gp,_a - .MN.gptop+28),r0 ret;;</pre>
10サイクル 12バイト	7サイクル 5バイト

【図 4 3】

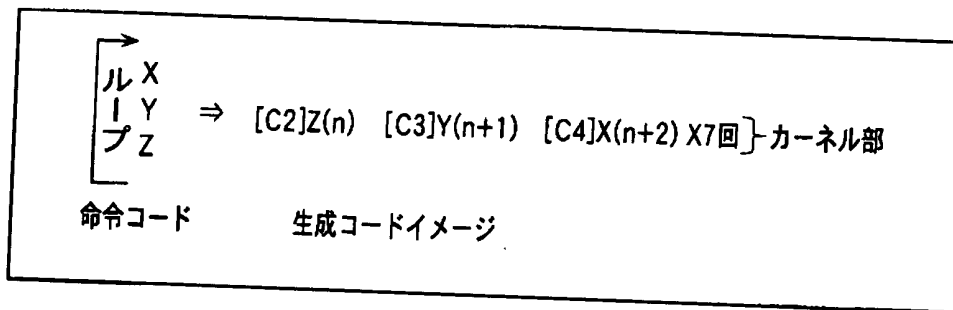
(a)



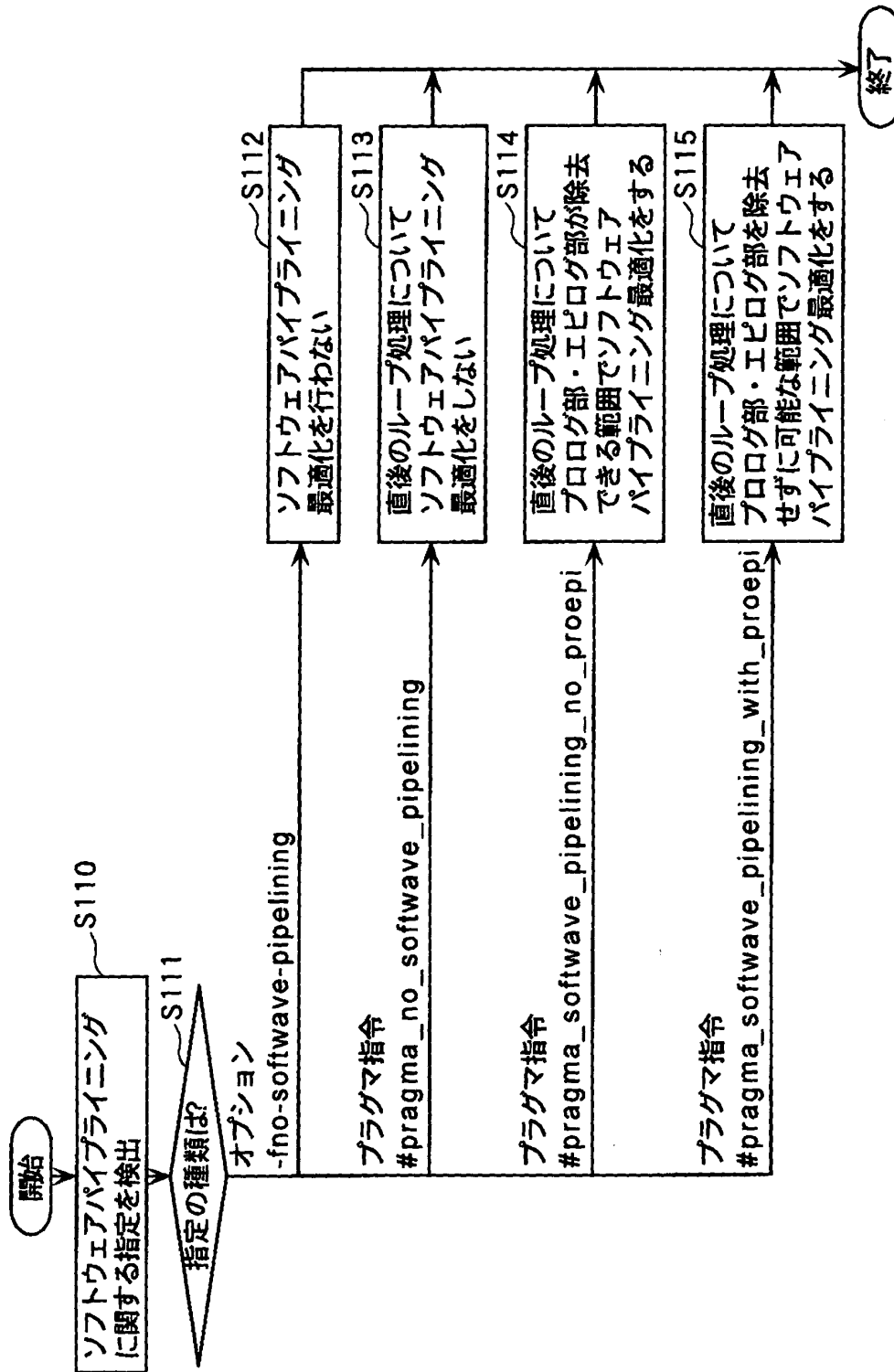
(b)



(c)



【図 44】



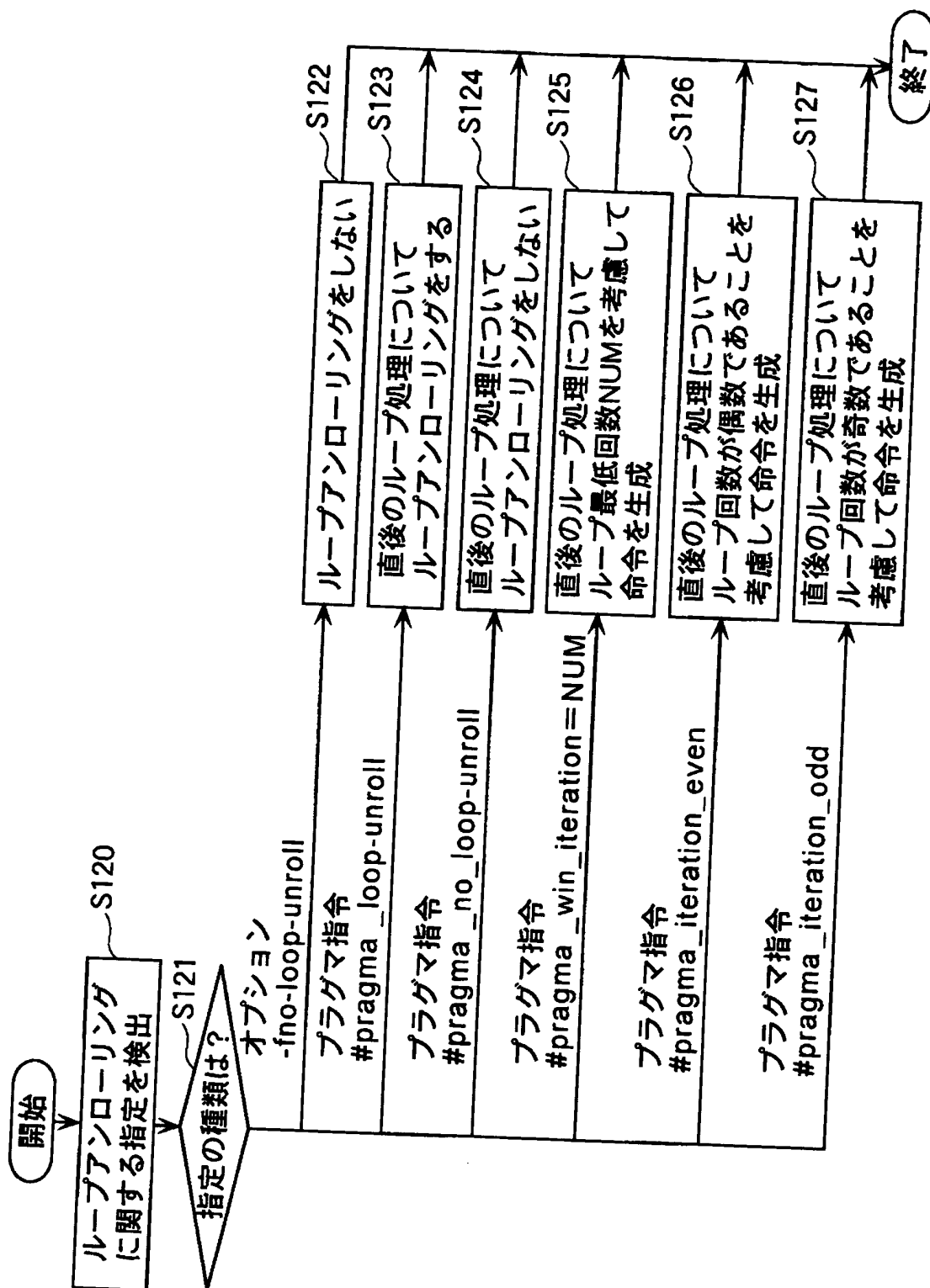
【図 45】

## コンパイルオプション-Oでのコンパイル

デフォルト	#pragma _software_pipelining_with_proepi 指令
<pre> inta[100]; intb[100]; voidsample(int c) {     int i;      for(i=0; i&lt;100; i++) {         a[i] = b[i] + c;     } } </pre>	<pre> int a[100]; int b[100]; void sample(int c) {     int i;     #pragma _software_pipelining_with_proepi     for(i=0; i&lt;100; i++) {         a[i] = b[i] + c;     } } </pre>
<pre> ld r5,(gp,_x\$ - .MN.gptop);;  mov r1,98 settar C6,C4,L00030 ld r4,(gp,_v\$ - .MN.gptop);; L00030 [C4] add r2,r3,r0 [C6] ld r3,(r5+);; [C4] st (r4+),r2 [C6] jloop C6,C4,tar,r1,-1;;  ret ;; </pre>	<pre> ld r5,(gp,_x\$ - .MN.gptop);; ld r4,(r5+);; mov r1,96 settar C6,L00023 ld r3,(gp,_v\$ - .MN.gptop);; L00023 add r2,r4,r0 ld r4,(r5+);; st (r3+),r2 [C6] jloop C6,C4,tar,r1,-1;; add r2,r4,r0;; st (r3+),r2 ret ;; </pre>
2+2×101+3=207サイクル 9バイト	3+2×99+3=204サイクル 12バイト



【図46】



【図 4 7】

#pragma \_loop\_unroll指令の実行速度向上例

デフォルト	#pragma _loop_unroll指令
<pre> int a[100]; int b[100]; void sample(int c) {     int i, ret=0;      for(i=0; i&lt;100; i++) {         ret += a[i] * a[i];     }     return ret; } </pre>	<pre> int a[100]; int b[100]; void sample(int c) {     int i, ret=0;     #pragma _loop_unroll     for(i=0; i&lt;100; i++) {         ret += a[i] * a[i];     }     return ret; } </pre>
<pre> mov r0,0 mov r1,98;; settar C6,C4,L00013  ld r3,(gp,_a\$ - .MN.gptop) mul m0,r2,gp,0;; L00013 [C4] mac m0,r0,r2,r2,m0  [C6] ld r2,(r3+)  [C6] jloop C6,C4,tar,r1,-1;; ret ;; </pre>	<pre> ld r3,(gp,_a\$ - .MN.gptop);; mov r0,0 mov r1,48;; settar C6,C2:C4,L00013 add r5,r3,4  mul m0,r2,gp,0;; L00013 [C2] mac m0,r0,r2,r2,m0 [C3] add r3,r3,8 [C3] ld r2,(r5+)8;; [C3] mac m0,r0,r4,r4,m0 [C4] ld r4,(r3) [C6] jloop C6,C2:C4,tar,r1,-1;; ret ;; </pre>
<p>2+2×101+3=207サイクル 9バイト</p>	<p>3+2×52+3=110サイクル 13バイト</p>

【図48】

デフォルト	#pragma _loop_unroll指令 と #pragma _align_local_pointer指令
<pre> int a[100]; int b[100]; void sample(int c) {     int i;      int *pa=a;     int *pb=b;      for(i=0; i&lt;100; i++) {         *pa++ = i * c &gt;&gt; *pb++;     } } </pre>	<pre> int a[100]; int b[100]; void sample(int c) {     int i;     #pragma _align_local_pointer=8 pa,pb     int *pa=a;     int *pb=b;     #pragma _loop_unroll     for(i=0; i&lt;100; i++) {         *pa++ = i * c &gt;&gt; *pb++;     } } </pre>
<pre> mov r4,0 ld r6,(gp,_a\$ - .MN.gptop);; mov r1,98 settar C6,C4,L00025 ld r5,(gp,_b\$ - .MN.gptop);; L00025 [C4] asr r2,r4,r3;;  [C6] ld r3,(r5+);; [C4] add r4,r4,r0 [C4] st (r6+),r2 [C6] jloop C6,C4,tar,r1,-1;; ret ;; </pre>	<pre> mov r6,0 ld r8,(gp,_a\$ - .MN.gptop);; mov r1,48 settar C6,C4,L00016 ld r7,(gp,_b\$ - .MN.gptop);; L00016 [C2] add r6,r6,r0;; [C2] stp (r8+),r2:r3 [C3] asr r2,r6,r4 [C3] add r6,r6,r0;; [C3] asr r3,r6,r5 [C4] ldp r4:r5,(r7+) [C6] jloop C6,C2:C4,tar,r1,-1;; ret ;; </pre>
<p>2+3×101+3=308サイクル 11バイト</p>	<p>2+3×51+3=158サイクル 13バイト</p>

【図 4 9】

#pragma _min_iteration 指令なし	#pragma _min_iteration 指令あり
<pre> int a[101]; int b[101]; void sample(int c, int end) {     int i;     int *pa=a;     int *pb=b;      for (i = 0; i &lt;end; i++) {         *pa++ = i * c &gt;&gt; *pb++;     }     *pa = end; } </pre>	<pre> int a[101]; int b[101]; void sample(int c, int end) {     int i;     int *pa=a;     int *pb=b;      #pragma _min_iteration=4     for (i = 0; i &lt;end; i++) {         *pa++ = i * c &gt;&gt; *pb++;     }     *pa = end; } </pre>
<pre> cmple C0,r1,0 ld r5,(gp,_a\$ - .MN.gptop);; mov r4,0 ld r6,(gp,_b\$ - .MN.gptop) [C0] br L00016;; mov r3,0 settar L00017 L00017 ld r2,(r6+);; add r4,1;; cmplt C0,r4,r1 asr r2,r3,r2;; add r3,r3,r0 st (r5+),r2 [C0] jmpf tar;; L00016 st (r5),r1 ret ;; </pre>	<pre> mov r5,0 ld r7,(gp,_a\$ - .MN.gptop);; settar C6,C4,L00027 sub r2,r1,2 ld r6,(gp,_b\$ - .MN.gptop);;  L00027 [C4] asr r3,r5,r4;; [C6] ld r4,(r6+);; [C4] add r5,r5,r0 [C4] st (r7+),r3 [C6] jloop C6,C4,tar,r2,-1;;  st (r7),r1 ret ;; </pre>
2+4×100+3=405 サイクル 16 バイト	2+3×101+3=308 サイクル 12 バイト

【図 5 0】

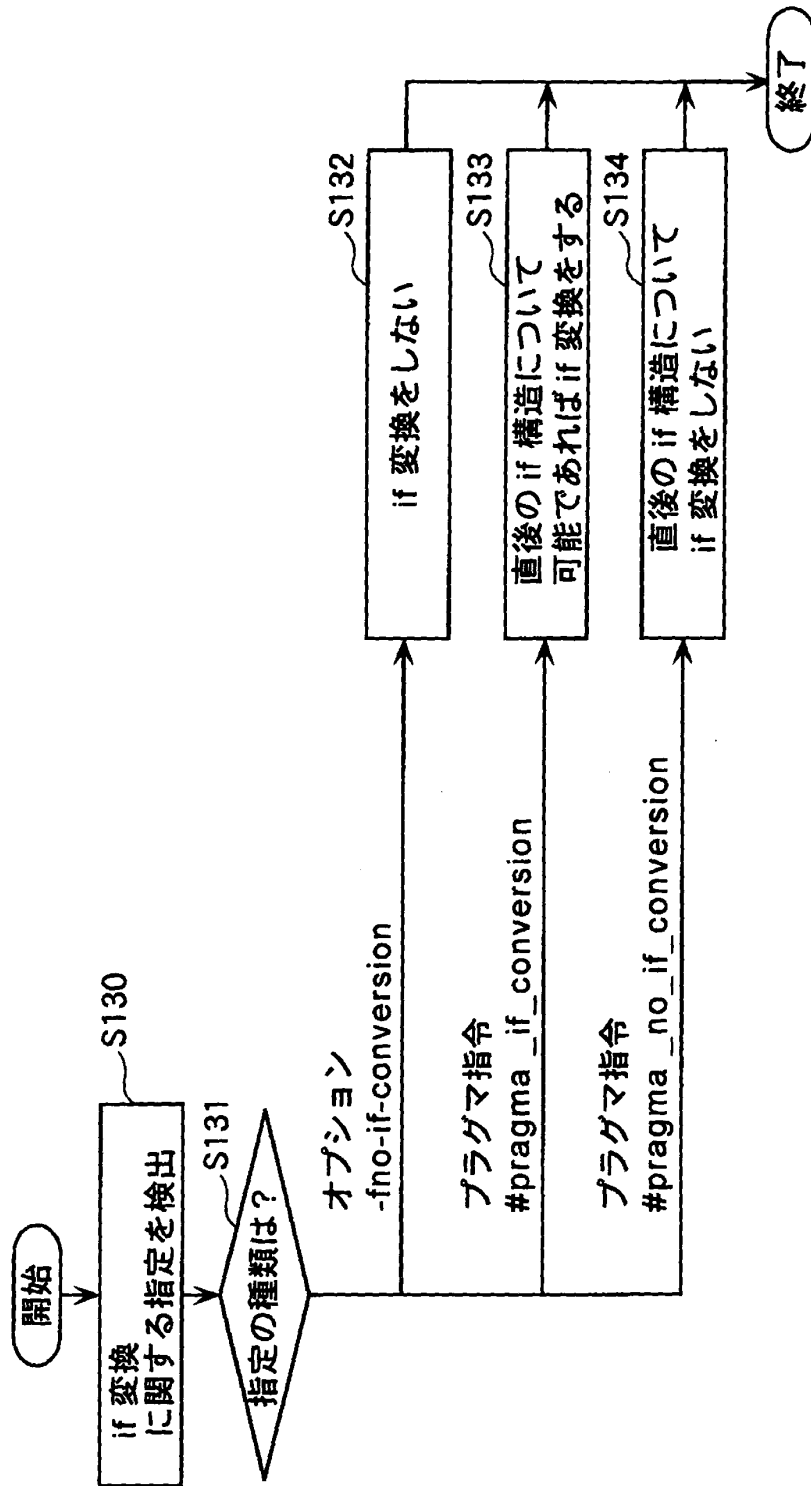
## ループアンローリング最適化が不可能な例

ループ回数が不明なC言語ソース	コンパイル結果
<pre> void sample(int c, int end) {     int i;     #pragma _align_local_pointer=8 pa, pb;     int *pa=a;     int *pb=b;     #pragma _loop_unroll     #pragma _min_iteration=4     for (i = 0; i &lt;end; i++) {         *pa++ = i * c &gt;&gt; *pb++;     }     *pa = end; } </pre>	<pre> mov r5,0 ld r7,(gp,_a\$ - .MN.gptop);; settar C6,C4,L00026 sub r2,r1,2 ld r6,(gp,_b\$ - .MN.gptop);; L00026 [C4] asr r3,r5,r4;; [C6] ld r4,(r6+);; [C4] add r5,r5,r0 [C4] st (r7+),r3 [C6] jloop C6,C4,tar,r2,-1;; st (r7),r1 ret ;; </pre>

【图 5 1】

#pragma_iteration_even指令	#pragma_iteration_odd指令
<pre> int a[101]; int b[101]; void sample(int c, int end) {     int i;     #pragma_align_local_pointer=8 pa, pb     int *pa=a;     int *pb=b;     #pragma_min_iteration=50     #pragma_loop_unroll     #pragma_iteration_even     for (i = 0; i &lt;end; i++) {         *pa++ = i * c &gt;&gt; *pb++;     }     *pa = end; } </pre>	<pre> int a[101]; int b[101]; void sample(int c, int end) {     int i;     #pragma_align_local_pointer=8 pa, pb     int *pa=a;     int *pb=b;     #pragma_min_iteration=50     #pragma_loop_unroll     #pragma_iteration_odd     for (i = 0; i &lt;end; i++) {         *pa++ = i * c &gt;&gt; *pb++;     }     *pa = end; } </pre>
<pre> sub r2,r1,6;; mov r6,0 asr r2,1 ld r7,(gp,_a\$ - .MN.gptop);; settar C6,C4,L00036 add r2,1 ld r8,(gp,_b\$ - .MN.gptop);; L00036 [C4] asr r3,r6,r4;; [C4] add r6,r6,r0 [C4] st (r7+),r3;; [C4] asr r3,r6,r5;; [C6] ldp r4:r5,(r8+);; [C4] add r6,r6,r0 [C4] st (r7+),r3 [C6] jloop C6,C4,tar,r2,-1;;  st (r7),r1 ret ;; </pre>	<pre> sub r2,r1,7;; mov r6,0 asr r2,1 ld r7,(gp,_a\$ - .MN.gptop);; settar C6,C4,L00046 add r2,1 ld r8,(gp,_b\$ - .MN.gptop);; L00046 [C4] asr r3,r6,r4;; [C4] add r6,r6,r0 [C4] st (r7+),r3;; [C4] asr r3,r6,r5;; [C6] ldp r4:r5,(r8+);; [C4] add r6,r6,r0 [C4] st (r7+),r3 [C6] jloop C6,C4,tar,r2,-1;; ld r2,(r8+);; asr r2,r6,r2;; st (r7+),r2;; st (r7),r1 ret ;; </pre>

【図 52】

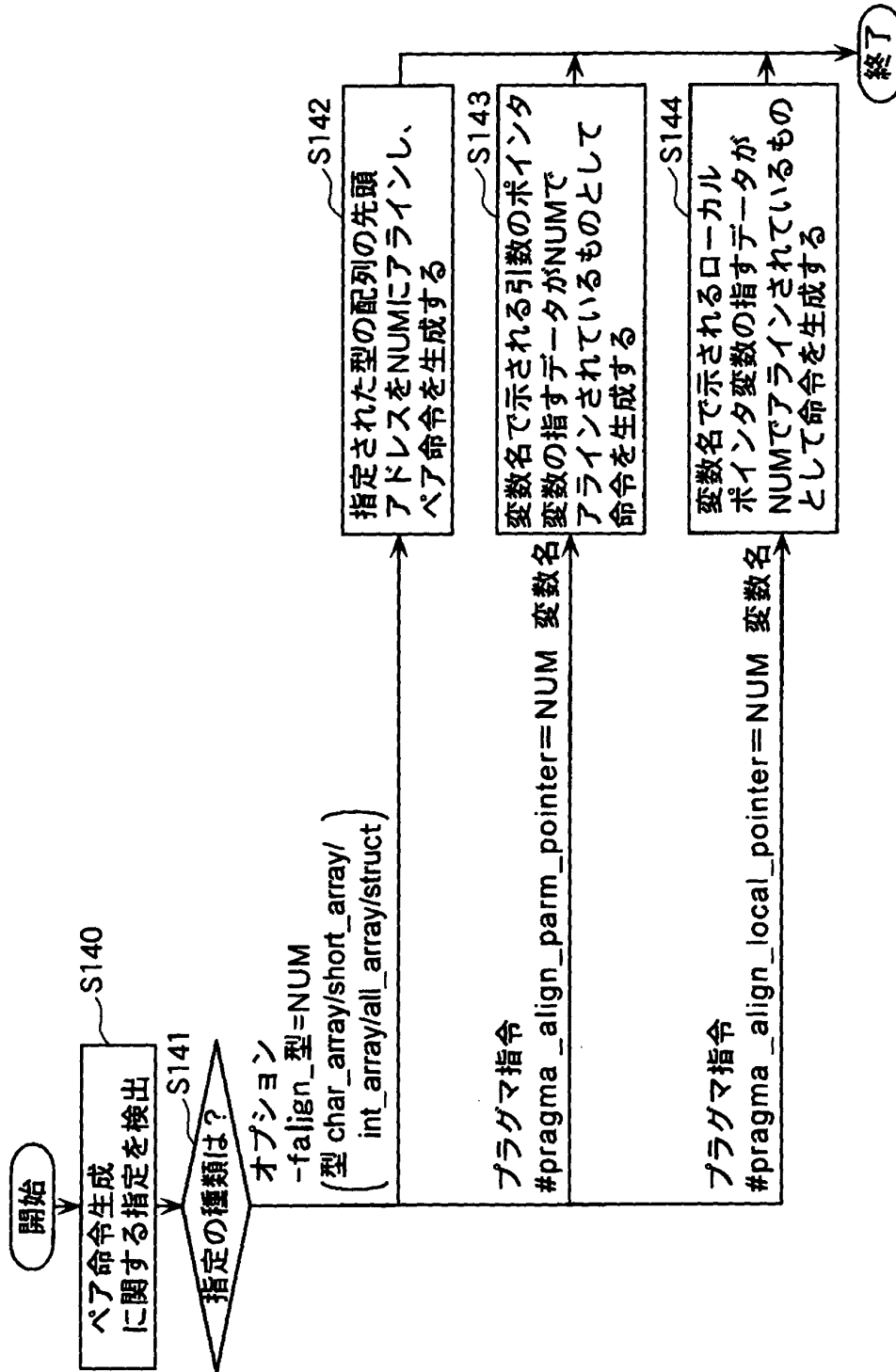


【図53】

#pragma_no_if_conversion指令	#pragma_no_if_conversion指令
<pre> int func(int a) {     int b;     #pragma_no_if_conversion     if(a == 2)         b = 0;     else         b = 1;     return b; } </pre>	<pre> int func(int a) {     int b;     #pragma_if_conversion     if(a == 2)         b = 0;     else         b = 1;     return b; } </pre>
<pre> cmpne C0,r0,2 ;; [C0] br L00002 ;; //分岐が生成されます     mov r0,0     ret ;; L00002     mov r0,1     ret ;; </pre>	<pre> cmpeq C0:C1,r0,2 ;; [C0] mov r0,0 //分岐命令のかわりに、 [C1] mov r0,1 //条件付き実行命令が生成されます     ret ;; </pre>
then側5サイクル, else側7サイクル 12バイト;	4サイクル 8バイト



【図 5 4】



【図 55】

「アライン指定なし」	「オプション'-falign-short-array=4指定」
<pre> short src[8]; short dst[8];  void align1(void) {     char dummy;     int i;      for(i=0; i&lt;8; i+=2)         dst[i] = src[i] + src[i+1]; } </pre>	<p>変更なし</p>
<pre> addu r5, gp_src - .MN.gptop addu r4, gp_src - .MN.gptop+2 ;; addu r3, gp_dst - .MN.gptop mov r0, 2 settar C6, C4, L00005 ;; L00005 [C4] ldh r1, (r4)+4 ;; //アラインメント不明のため、 [C4] add r1, r2, r1 // 2つデータは独立して [C6] ldh r2, (r5)+4 ;; //ロードされます(ldh命令) [C4] sth (r3)+4, r1 [C6] jloop C6, C4, tar, r0, -1 ;; //L00005 ret ;; </pre>	<pre> addu r5, gp_src - .MN.gptop ;; addu r4, gp_dst - .MN.gptop mov r0, 2 settar C6, C4, L00005 ;; L00005 [C4] add r1, r2, r3 //アラインメント4のため、 [C6] ldhp r2, r3, (r5+) ;; //ベアアクセス命令(ldhp) [C4] sth (r4)+4, r1 //が生成されます。 [C6] jloop C6, C4, tar, r0, -1 ;; //L00005 ret ;; </pre>
<p>2+4×5+3= 25サイクル 22バイト</p>	<p>2+2×5+3=15サイクル 18バイト</p>

【図 56】

「アライン指定なし」	「#pragma _align_parm_pointer 指令使用」
<pre> int align2(short * src) {     int    v = 0 ;     for(int i=0 ; i&lt;100 ; i+=2){         v += src[i] * src[i+1] ;     }     return v ; } </pre>	<pre> #pragma _align_parm_pointer=4 src int align2(short * src) {     int    v = 0 ;     for(int i=0 ; i&lt;100 ; i+=2){         v += src[i] * src[i+1] ;     }     return v ; } </pre>
<pre> mov r5,0 mov r1,48 settarC6,C4,L00005 ;; mov r4,r0 add r3,r0,2 mul m0,r2,gp,0 ;; L00005 [C4] ldh r0,(r3+)4 ;; //ソフトウェアバイパス適用 [C4] lmac m0,r5,r2,r0,m0 //データのアラインメント不明のため [C6] ldh r2,(r4+)4 //ldh命令を2回実行 [C6] jloop C6,C4,tar,r1,-1 ;; //L00005 mov r0,r5 ret ;; </pre>	<pre> mov r4,0 ;; mov r1,48 settarC6,C4,L00005 mul m0,r2,gp,0 ;; L00005 [C4] lmac m0,r4,r2,r3,m0 //ソフトウェアバイパス適用 [C6] kdhp r2:r3,(r0+) //ベアアクセス命令使用可能。 [C6] jloop C6,C4,tar,r1,-1 ;; //L00005 mov r0,r4 ret ;; </pre>
<p>2+3×51+3= 160サイクル 24バイト</p>	<p>2+2×51+3= 107サイクル 18バイト</p>

【図 57】

「アライン指令なし」	「#pragma _align_local_pointer 指令使用」
<pre> void align3(int n) {     short * from;     short * to;     int i;      from = &amp;(src[n]); to = &amp;(dst[n]);     for(i=0; i&lt;16; i++, from+=2, to+=2){         * to = * from;         * (to+1) = * (from+1);     } } </pre>	<pre> void align3(int n) {     # pragma _align_local_pointer=4 from,to     short * from;     short * to;     int i;      from = &amp;(src[n]); to = &amp;(dst[n]);     for(i=0; i&lt;16; i++, from+=2, to+=2){         * to = * from;         * (to+1) = * (from+1);     } } </pre>
<pre> ld r3,(gp_src\$ - .MN.gptop) ;; ld r2,(gp_dst\$ - .MN.gptop) ;; add r1,r0,r0 ;; mov r0,13 add r6,r3,r1 add r5,r2,r1 ;; settar C6,L00016 add r4,r5,2 add r3,r6,2 ;; L00016 ldh r1,(r6+)4 ;; ldh r2,(r3+)4 ;; sth (r5+)4,r1 ;; sth (r4+)4,r2 [C6] jloop C6,tar,r0,-1 ;; //L00016 ret ;; </pre>	<pre> ld r3,(gp_src\$ - .MN.gptop) ;; ld r2,(gp_dst\$ - .MN.gptop) ;; mov r1,13 add r0,r0,r0 ;; settar C6,L00016 add r4,r3,r0 add r0,r2,r0 ;; L00016 ldhp r2:r3,(r4+) ;; sth (r0+),r2:r3 [C6] jloop C6,tar,r1,-1 ;; //L00016 ret ;; </pre>
<p>5+4×16+3=72サイクル 30バイト</p>	<p>5+3×16+3=56サイクル 22バイト</p>

【書類名】            要約書

【要約】

【課題】    コンパイラによる最適化をユーザが緻密に制御することが可能な柔軟性の高いコンパイラを提供する。

【解決手段】    コンパイラ 1 0 0 に対するユーザからの指示（オプション及びプラグマ）を検出する解析部 1 1 0 と、解析部 1 1 0 からの指示等に従って、ユーザによるオプション及びプラグマによって指定された個別的な最適化処理を行う処理部（グローバル領域割り付け部 1 2 1、ソフトウェアパイプライン部 1 2 2、ループアンローリング部 1 2 3、if 変換部 1 2 4 及びペア命令生成部 1 2 5）からなる最適化部 1 2 0 等を備え、グローバル領域割り付け部 1 2 1 は、グローバル領域に配置する変数の最大データサイズの指定、グローバル領域に配置させる変数の指定、及び、グローバル領域に配置させない変数の指定に関するオプション及びプラグマに従った最適化処理を行う。

【選択図】            図 1

認定・付加情報

特許出願の番号	特願2002-195305
受付番号	50200977716
書類名	特許願
担当官	第七担当上席
作成日	平成14年 7月 4日

<認定情報・付加情報>

【提出日】	平成14年 7月 3日
-------	-------------

出 願 人 履 歴 情 報

識別番号

[000005821]

1. 変更年月日

1990年 8月28日

[変更理由]

新規登録

住 所

大阪府門真市大字門真1006番地

氏 名

松下電器産業株式会社